On parallelizing dual decomposition in stochastic integer programming

Miles Lubin[1,a], Kipp Martin[b], Cosmin Petra[a], Burhaneddin Sandıkçı[b]

[a] *Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA*
[b] *University of Chicago Booth School of Business, Chicago, IL, USA*

Abstract

For stochastic mixed-integer programs, we revisit the dual decomposition algorithm of Carøe and Schultz from a computational perspective with the aim of its parallelization. We address an important bottleneck of parallel execution by identifying a formulation that permits the parallel solution of the *master* program by using structure-exploiting interior-point solvers. Our results demonstrate the potential for parallel speedup and the importance of regularization (stabilization) in the dual optimization. Load imbalance is identified as a remaining barrier to parallel scalability.

Keywords: stochastic programming, mixed-integer programming, column generation, dual decomposition, parallel computing, bundle methods

1. Introduction

Stochastic mixed-integer programming (SMIP) models with recourse [1] are commonly used in practice for making discrete decisions under uncertainty. Such models arise in applications in energy, routing, scheduling, production planning, and others, where parts or all of the data for the model are not completely known at the time decisions must be made, but can be approximated by some presumed stochastic model.

Although many practical instances remain difficult to solve, significant progress has been made in developing algorithms to solve these problems, particularly those with special structure such as pure-integer recourse or pure-binary first-stage decisions (for reviews, see [1, 2]). For more general SMIP problems, Sen [2] suggests the *dual decomposition* (DD) approach of Carøe and Schultz [3] or the *branch-and-price* (BP) approach of Lulli and Sen [4]. This paper focuses on these two approaches from the perspective of parallel computing.

In our theoretical development in §2, we demonstrate an effective equivalence between the nonsmooth Lagrangian dual problem solved by DD and the restricted master problem solved by BP. While it was previously known that these problems have the same optimal values, the effective equivalence is stronger in that solving one provides an optimal solution to both. This fact relates to the so-called primal-recovery properties of subgradient approaches applied to Lagrangian duals, which only recently have become more widely known in the optimization community [5, 6]. Both approaches are therefore seen to solve the same relaxation simply by different algorithms for nonsmooth optimization, the former by the proximal bundle method [7] and the latter by a more slowly convergent cutting-plane method, as discussed in §3.

In §4, we analyze our new formulation for the quadratic program (QP) master problem of the proximal bundle method, considering the particular structure induced by relaxing nonanticipativity constraints. The new formulation results in a block-angular QP that can be solved efficiently by recently developed interior-point solvers for structured QPs. Improvements of several orders of magnitude are observed over the QP solvers from off-the-shelf proximal bundle codes.

Reducing the time spent solving the master problem significantly increases the scope for parallelism, which has previously been identified but not exploited in an implementation. In §5, we present our numerical results from a preliminary parallel implementation on a high-performance cluster.

2. Dual decomposition and branch-and-price

Consider the following two-stage SMIP with recourse:

$$z = \min\left\{c^\top x + Q(x) : \ \ Ax \le b, \ \ x \in X\right\}, \tag{1}$$

where $Q(x) = \mathbb{E}_\xi\left[\min\{q(\xi)^\top y : \ \ Wy \le h(\xi) - T(\xi)x, \ \ y \in Y\}\right]$. The parameters $c \in \mathbb{Q}^{n_1}$, $b \in \mathbb{Q}$, $A \in \mathbb{Q}^{m_1\times n_1}$, and $W \in \mathbb{Q}^{m_2\times n_2}$ are fixed and known. The vector ξ is a random variable, assumed here to have a discrete distribution with r possible realizations $\xi_1, \dots, \xi_r$ and corresponding probabilities $p_1, \dots, p_r$. Realization $j = 1, \dots, r$, known as *scenario j*, contains the rational data $\big(q(\xi_j), h(\xi_j), T(\xi_j)\big)$, now (q_j, h_j, T_j) for brevity, where the vectors q_j and h_j and the matrix T_j have conformable dimensions. The sets $X \subseteq \mathbb{R}^{n_1}_+$ and $Y \subseteq \mathbb{R}^{n_2}_+$ denote other restrictions, possibly including integer or binary constraints, on the decision variables x and y, respectively. For $j = 1, \dots, r$, define the set

$$S_j := \left\{(x, y_j) : \ \ Ax \le b, \ \ T_j x + W y_j \le h_j, \ \ x \in X, \ \ y_j \in Y\right\}.$$

Email addresses: mlubin@mit.edu (Miles Lubin[1,]), kmartin@chicagobooth.edu (Kipp Martin), petra@mcs.anl.gov (Cosmin Petra), burhan@chicagobooth.edu (Burhaneddin Sandıkçı)

[1]Present affiliation: MIT Operations Research Center, Cambridge, MA, USA

The *deterministic equivalent* problem to (1), assumed to be feasible and bounded, is

$$z = \min\left\{c^\top x + \textstyle\sum_{j=1}^r p_j q_j^\top y_j : \ (x, y_j) \in S_j, j = 1, \ldots, r\right\}. \quad (2)$$

Also consider the equivalent *split-variable* formulation:

$$\begin{aligned} z = \min\{ & \textstyle\sum_{j=1}^r p_j(c^\top x_j + q_j^\top y_j) : \\ & (x_j, y_j) \in S_j, \ \ j = 1, \ldots, r, \ \ x_{\cdot} = x_1 = \ldots = x_r\}. \end{aligned} \quad (3)$$

The constraints $x_{\cdot} = x_1 = \ldots = x_r$ are known as the nonanticipativity conditions, which force the first-stage decision x to be the same under each scenario. We have introduced an additional variable $x_{\cdot}$ and modeled nonanticipativity as

$$x_j - x_{\cdot} = 0, \ \ j = 1, \ldots, r. \quad (4)$$

This representation of nonanticipativity, as used by Lulli and Sen [4], differs from the one used by Carøe and Schultz [3], who instead represent it by a set of equalities solely on the variables $x_1, \ldots, x_r$ of the form $\sum_{j=1}^r H_j x_j = 0$. (For example, $x_1 - x_j = 0, \ j = 2, \ldots, r$.) The representations are equivalent; however, as shown later (4) is advantageous for computation.

Relaxing the nonanticipativity constraints, one may write the Lagrangian relaxation of the split-variable formulation as

$$\begin{aligned} D(\lambda_1, \ldots, \lambda_r) = \min\{ & \textstyle\sum_{j=1}^r [L_j(x_j, y_j, \lambda_j) - \lambda_j^\top x_{\cdot}] : \\ & (x_j, y_j) \in S_j, \ \ j = 1, \ldots, r\}, \end{aligned} \quad (5)$$

where $L_j(x_j, y_j, \lambda_j) = p_j(c^\top x_j + q_j^\top y_j) + \lambda_j^\top x_j$ for $j = 1, \ldots, r$. As $x_{\cdot}$ is unconstrained, the condition $\sum_{j=1}^r \lambda_j = 0$ is required for boundedness of the Lagrangian. With this condition, the $\lambda_j^\top x_{\cdot}$ terms vanish, and the Lagrangian is separable into $D(\lambda_1, \ldots, \lambda_r) = \sum_{j=1}^r D_j(\lambda_j)$, where, for $j = 1, \ldots, r$,

$$D_j(\lambda_j) = \min_{x_j, y_j}\left\{L_j(x_j, y_j, \lambda_j) : \ (x_j, y_j) \in S_j\right\}. \quad (6)$$

For any choice of $\lambda_1, \ldots, \lambda_r$, it is clear that $D(\lambda_1, \ldots, \lambda_r) \le z$; that is, the Lagrangian relaxation provides a valid lower bound on the optimal value z of (1). A natural problem is then to find the best such bound. This is known as the Lagrangian dual problem, and is expressed as [2]

$$z_{\text{LD}} = \max_{\lambda_1, \ldots, \lambda_r}\left\{\textstyle\sum_{j=1}^r D_j(\lambda_j) : \sum_{j=1}^r \lambda_j = 0\right\}. \quad (7)$$

Because of the potential nonconvexity introduced by the sets X and Y, the optimal value z_{LD} is typically, but not always, strictly less than z. We restate Proposition 2 of [3], which provides a characterization of the optimal value.

Proposition 1. *The optimal value z_{LD} of the Lagrangian dual* (7) *equals the optimal value of the linear program*

$$\min\{\textstyle\sum_{j=1}^r p_j(c^\top x_j + q_j^\top y_j) : \ (x_j, y_j) \in \text{conv}(S_j), \ x_j = x_{\cdot}, \forall\, j\}. \quad (8)$$

The Lagrangian dual (7) is a concave, nonsmooth optimization problem, which Carøe and Schultz [3] propose to solve with subgradient methods. The bounds generated from the Lagrangian dual are then used within a branch-and-bound procedure. This is the so-called dual decomposition (DD) approach. Lulli and Sen [4], on the other hand, propose to solve (8) directly using a column generation procedure as part of a branch-and-price (BP) algorithm.

Below we present the classical subgradient cutting-plane approach and then demonstrate that applying the cutting-plane method to (7) is in fact equivalent to solving (8) by column generation. In particular, it is easy to recover a primal solution to (8) from DD. This primal solution can then be used as in BP.

Each $D_j(\lambda_j)$ is concave in λ_j, and γ_j^k is a subgradient of $D_j(\lambda_j)$ at the point λ_j^k, if

$$D_j(\lambda_j) \le D_j(\lambda_j^k) + (\gamma_j^k)^\top(\lambda_j - \lambda_j^k)$$

for all λ_j. Since $L_j(x_j, y_j, \lambda_j) = p_j(c^\top x_j + q_j^\top y_j) + \lambda_j^\top x_j$, given a λ_j^k, the corresponding subgradient γ_j^k is equal to x_j^k, where (x_j^k, y_j^k) is a solution to (6). The cutting-plane method (as described in Figure 1) replaces each $D_j(\lambda_j)$ with a relaxation using a set of subgradients and solves the following linear program (LP) at each iteration:

$$\begin{aligned} \max \quad & \textstyle\sum_{j=1}^r \theta_j && (9) \\ \text{s.t.} \quad & \textstyle\sum_{j=1}^r \lambda_j = 0, && (x_{\cdot}) \\ & \theta_j \le D_j(\lambda_j^k) + (x_j^k)^\top(\lambda_j - \lambda_j^k), \quad {\scriptstyle j=1,\ldots,r,\ k=1,\ldots,K.} && (z_j^k) \end{aligned}$$

```
Initialize: Choose a relative convergence tolerance $\epsilon$.
            $K \leftarrow 1$, $\lambda_j^K \leftarrow 0$ for $j = 1, \ldots, r$.
            Solve (6) for $j = 1, \ldots, r$, saving optimal value
            $D_j(\lambda_j^K)$ and solution $x_j^K$.
Step 1:     Solve (9), saving optimal $\theta_j^*$ and $\lambda_j^*$ for all $j$.
Step 2:     $K \leftarrow K + 1$, $\lambda_j^K \leftarrow \lambda_j^*$ for $j = 1, \ldots, r$.
            Solve (6) for $j = 1, \ldots, r$, saving optimal value
            $D_j(\lambda_j^K)$ and solution $x_j^K$.
Step 3:     If $\sum_j[\theta_j^* - D_j(\lambda_j^K)]/[1 + |\sum_j D_j(\lambda_j^K)|] < \epsilon$ terminate;
            else add $D_j(\lambda_j^K) + (x_j^K)^\top(\lambda_j - \lambda_j^K)$ to (9).
Step 4:     Goto Step 1
```

Figure 1: Pseudocode for cutting-plane algorithm.

It is easy to recover a primal solution to (8) from the LP solution of (9). The dual of (9) is

$$\begin{aligned} \min \quad & \textstyle\sum_{j=1}^r \sum_{k=1}^K \left[D_j(\lambda_j^k) - (x_j^k)^\top \lambda_j^k\right] z_j^k && (10) \\ \text{s.t.} \quad & \textstyle\sum_{k=1}^K z_j^k = 1, \quad j = 1, \ldots, r, && (11) \\ & \textstyle\sum_{k=1}^K z_j^k x_j^k = x_{\cdot}, \quad j = 1, \ldots, r, && (12) \\ & z_j^k \ge 0, \quad j = 1, \ldots, r, \quad k = 1, \ldots, K. && (13) \end{aligned}$$

For a given λ_j^k, let (x_j^k, y_j^k) be the corresponding optimizer in (6). Minimizing a linear function over the feasible region S_j of a mixed-integer linear program is equivalent to minimizing over the convex hull $\text{conv}(S_j)$, of the feasible region. Therefore, $D_j(\lambda_j) = \min_{x_j, y_j}\{L_j(x_j, y_j, \lambda_j) : \ (x_j, y_j) \in \text{conv}(S_j)\}$. Then the definitions of $D_j(\lambda_j^k)$ and $L_j(x_j^k, y_j^k, \lambda_j^k)$ imply that the objective function (10) is equal to

$$\textstyle\sum_{j=1}^{r}\sum_{k=1}^{K}\left[(p_j(c^\top x_j^k + q_j^\top y_j^k) + (x_j^k)^\top \lambda_j^k) - (x_j^k)^\top \lambda_j^k\right] z_j^k = \sum_{j=1}^{r}\sum_{k=1}^{K} p_j(c^\top x_j^k + q_j^\top y_j^k) z_j^k,$$

which is precisely the objective of the standard restricted master for (8), as described in Lulli and Sen [4]. Hence, feasible (and at convergence, optimal) solutions to (8) are obtained from the dual solution of (9).

Indeed, by using nearly any subgradient-based method to solve (7), one may obtain at a minimal cost an optimal solution to (8). This fact, while well known to specialists in the general case (see, e.g., [6, 8]), has only recently come to attention in the context of Lagrangian relaxation in integer programming [5]. To the best of our knowledge, this result has not been stated in the context of DD and BP.

The branch-and-bound algorithm used in DD calls for branching on *disagreements* in the primal solutions x_j^k, $j = 1, \dots, r$, produced by the subproblems, whereas in BP the solution to (8) is used for branching decisions, similar to how the solution to the LP relaxation is used in classical branch and bound for integer programs. Hence, while known to use the same relaxation (8) in a theoretical sense, DD and BP have been viewed as computationally different approaches [2, 9]; we have demonstrated a closer computational connection than previously observed. Frangioni [5] suggests using both the solution to the convexification (8) and the primal solutions to the subproblems within branch and bound. We leave the exploration of this possibility in the present context for future research.

3. Improvements to the cutting-plane algorithm

The cutting-plane algorithm is known to be unstable and to converge slowly on practical instances [6, 7]. Modern approaches apply some form of regularization to the standard cutting-plane approach, potentially resulting in a more difficult master program but also providing a significant reduction in the total number of iterations required. In particular, the proximal bundle method [7] uses a quadratic penalty in the objective to regulate the step length at each iteration. This approach has appeared in the stochastic programming literature as Ruszczyński's regularized decomposition [10]. Other approaches include ℓ_∞ trust regions and level regularization [11]; these have been used in stochastic programming, for example, by [12] and [13], respectively. The relative performance of different forms of regularization is generally not well understood and is typically problem dependent [14, 15].

We focus on the proximal bundle method, which is the most widely used regularization method. In this variant, a quadratic penalty term $\sum_{j=1}^{r} \|\lambda_j - \lambda_j^+\|_2^2$ is subtracted from the objective function of (9), where $(\lambda_1^+, \lambda_2^+, \dots, \lambda_r^+)$ is the current "prox-center" with $\sum_{j=1}^{r} \lambda_j^+ = 0$. The modified master (9) is

$$\begin{aligned}
\max_{\theta,\beta}\quad & \textstyle\sum_{j=1}^{r} \theta_j - \frac{1}{2}\tau \sum_{j=1}^{r} \|\beta_j\|_2^2 && (14)\\
\text{s.t.}\quad & \textstyle\sum_{j=1}^{r} \beta_j = 0 && (w)\\
& \theta_j - (x_j^k)^\top \beta_j \le D_j(\lambda_j^k) + (x_j^k)^\top(\lambda_j^+ - \lambda_j^k), \quad {\scriptstyle j=1,\dots,r,\; k=1,\dots,K.} && (z_j^k)
\end{aligned}$$

where $\beta_j := \lambda_j - \lambda_j^+$. The regularization parameter τ is typically adjusted at each iteration; see [8].

It is generally advantageous to solve the Lagrangian dual of (14):

$$\min_{w,z} \quad \sum_{j=1}^{r}\left(\sum_{k=1}^{K} z_j^k \left(D_j(\lambda_j^k) + (x_j^k)^T(\lambda_j^+ - \lambda_j^k)\right) + \frac{1}{2\tau}\|w - \sum_{k=1}^{K} z_j^k x_j^k\|^2\right) \tag{15}$$

$$\text{s.t.} \quad \textstyle\sum_{k=1}^{K} z_j^k = 1, \quad z_j^k \ge 0, \qquad j = 1, \dots, r, \quad k = 1, \dots, K.$$

Using the solution of (15), one can recover the optimal β_j by $\beta_j = \frac{1}{\tau}\left(\sum_{k=1}^{K} z_j^k x_j^k - w\right)$. Since w is an n_1-vector and the z_j^k are scalars, (15) has $K \times r + n_1$ variables, which is typically significantly smaller than the $(n_1 + 1) \times r$ variables of (9) or (14). We also expect $K \times r >> n_1$. If this does not hold, it could be advantageous to eliminate w by noting that $w = \frac{1}{r}\sum_{j=1}^{r}\sum_{k=1}^{K} z_j^k x_j^k$ at optimality (this may be derived from the Karush-Kuhn-Tucker conditions). However, this elimination destroys the particular structure that is discussed in §4, and a general sparsity-exploiting solver can perform this elimination automatically.

From standard convergence results [7],

$$\|w - \textstyle\sum_{k=1}^{K} z_j^k x_j^k\|^2 \to 0, \; j = 1, \dots, r$$

at convergence of the proximal bundle method. Hence, in the limit, $w = \sum_{k=1}^{K} z_1^k x_1^k = \cdots = \sum_{k=1}^{K} z_r^k x_r^k$, which are precisely the constraints (12) of the standard column generation master. So, one recovers the solution to the convexification (8) directly as the optimal w at convergence of the proximal bundle method.

4. Parallel solution of the master program

It has been observed (e.g., [4]) that the cutting-plane algorithm described in Figure 1 exhibits scope for parallelism in Step 2, where r independent integer programs must be solved. The same observation holds for regularized variants discussed in §3. The potential for parallel speedup, however, is limited according to Amdahl's law [16] by the serial execution bottleneck of solving the master program, for example, the LP (9) or the QP (15). We address this bottleneck by identifying the scope for parallelism in solving the master itself.

Definition. A QP has *dual block-angular structure* if its constraint matrix and objective Hessian matrix can be permuted to the form

$$\begin{pmatrix} X & & & \\ X & X & & \\ \vdots & & \ddots & \\ X & & & X \end{pmatrix} \text{ and } \begin{pmatrix} X & X & \cdots & X \\ X & X & & \\ \vdots & & \ddots & \\ X & & & X \end{pmatrix},$$

respectively, with X's indicating the only (possibly) nonzero blocks.

A key observation is that the proximal bundle QP master (15), as well as the LP dual of the cutting-plane master (9), exhibit a dual block-angular structure. The Hessian of (15) exhibits the required structure by noting that for each j, the quadratic terms are $\sum_{i=1}^{n_1}(w_i - \sum_{k=1}^{K} z_j^k x_{ji}^k)^2$. Therefore, the only bilinear terms are $w_i z_j^k$, which correspond to the first n_1 rows of the Hessian, and $z_j^k z_j^{k'}, k \neq k'$, which correspond to the jth diagonal block. The w variables in fact do not appear in the constraints, and so the corresponding X blocks are entirely zero. This property of block-angular structure is a direct result of formulation (4); in particular, it does not hold if the nonanticipativity representation of Carøe and Schultz [3] is used.

Block-angular structure in linear and quadratic programs has been successfully exploited for parallelization within interior-point methods [17]. We follow this approach, a discussion of which is beyond the scope of this paper. Only minimal development, if any, is required to efficiently solve (15) using an existing structure-exploiting interior-point code.

While our analysis has been limited to two-stage formulations, we remark that the *nested* block-angular structure that could arise in the master of a multistage problem remains within the framework of parallel structure-exploiting interior-point methods [17].

Note that Kiwiel [18] developed a specialized active-set method for solving the QP master of the proximal bundle method for unstructured problems. However, this method cannot immediately accommodate the equality constraints of our formulation and it is unknown whether this approach could be successfully parallelized for block-angular structure.

5. Implementation and numerical results

In this section, we explore different computational aspects of dual decomposition, with a view toward parallel computation. We consider only the "root node"; no branching schemes were implemented. All experiments were performed on *Fusion*, a 320-node computing cluster at Argonne National Laboratory. Fusion has an InfiniBand QDR interconnect, and each node has two quad-core 2.6 GHz Xeon processors and 36 GB of RAM. Serial experiments were performed on a single node of Fusion.

We use publicly available two-stage SMIP instances. The `sslp` and `dcap` instances are available at `http://www2.isye.gatech.edu/~sahmed/siplib/`, and the `prod` instances are available at `http://people.orie.cornell.edu/huseyin/research/sp_datasets/sp_datasets.html`. Basic statistics about these instances are listed in Table 1. Note that all of the `sslp` and `dcap` instances have integer variables in both stages. The `prod` instances, on the other hand, are stochastic *linear* programs which will be of interest because of their relatively large number of first-stage variables. Further descriptions of the instances are available at the indicated websites. Because of space limitations, we report only on a subset of the instances available online.

Table 1: Test problem statistics.

Test	1st Stage			2nd-Stage Scenario		
Problem	Vars.	Intgr.	Cons.	Vars.	Intgr.	Cons.
`sslp_5_25`	5	5	1	130	125	30
`sslp_10_50`	10	10	1	510	500	60
`sslp_15_45`	15	15	1	690	675	60
`dcap233`	12	6	6	27	27	15
`dcap243`	12	6	6	36	36	18
`dcap332`	12	6	6	24	24	12
`dcap342`	12	6	6	32	32	14
`prod-small`	50	0	10	250	0	220
`prod-medium`	250	0	10	1400	0	250
`prod-large`	1,500	0	75	1,450	0	700

```
Initialize:  Choose a relative convergence tolerance $\epsilon$.
             $K \leftarrow 1, \tau \leftarrow 1, m = 0.1$, and $\lambda_j^+ \leftarrow 0$ for $j = \dots, r$.
             Solve (6) with $\lambda_j^+$ for all $j$, saving optimal solution $x_j^K$.
             $curObj \leftarrow \sum_j D_j(\lambda_j^+)$.
Step 1:      Solve (15), saving optimal $w^*, z_j^{k*}, \theta_j^*, \lambda_j^*$.
Step 2:      Let $v = (\sum_i \theta_i^*) - curObj$. If $v/(1 + |curObj|) < \epsilon$,
             terminate, else continue.
Step 3:      $K \leftarrow K + 1$.
Step 4:      Solve $D_j(\lambda_j^*)$ for $j = 1, \dots, r$, saving optimal value
             $D_j(\lambda_j^K)$ and solution $x_j^K$.
Step 5:      $newObj \leftarrow \sum_j D_j(\lambda_j^K)$. Let $u = 2\tau(1 - (newObj -
             curObj)/v)$.
Step 6:      Update $\tau \leftarrow \min(\max(u, \tau/10, 10^{-4}), 10\tau)$. (See [8])
Step 7:      If $(newObj - curObj > m \cdot v)$ update $\lambda_j^+ \leftarrow \lambda_j^*$,
             $curObj \leftarrow newObj$.
Step 8:      Goto Step 1.
```

Figure 2: Pseudocode of proximal bundle method as implemented. Note that we take a maximization view; some signs are inverted from typical statements of the algorithm, e.g., in [8]. An important mathematical feature of bundle methods is the ability to remove old subgradients ("compress the bundle") after Step 4; however, we do not implement this.

5.1. Serial experiments

The `sslp` and `dcap` instances are used to compare the performance of various methods, executed in serial, for optimizing the Lagrangian dual (7). We experiment with three methods: (*i*) the classical cutting-plane method (Figure 1), (*ii*) the proximal bundle method (Figure 2), and (*iii*) the ℓ_∞ trust-region method using the trust-region updating rules of Linderoth and Wright [12]. All algorithms use a relative convergence tolerance $\epsilon = 10^{-7}$, although the convergence criteria have slight mathematical differences. In our C++ implementation of the cutting-plane and ℓ_∞ trust-region methods, the master program, which is linear, is solved by `Clp` [19], hot-started by using the optimal basis from the previous solution. For the proximal bundle method, we use both our implementation and the off-the-shelf open-source implementation `ConicBundle` [20].

Within our implementation, we experiment with solving the QP (15) using a general sparsity-exploiting interior-point solver OOQP [21] (with the MA57 [22] sparse linear-algebra library) and then using the block-angular-structure-exploiting interior-point solver PIPS-IPM [23] (with LAPACK routines for dense

Table 2: Summary of results with serial experiments. All methods except `ConicBundle` were implemented by the authors. The cutting-plane method is shown to require many more iterations than regularized variants. Asterisk indicates exceeded time limit (7,200 seconds).

Instance			Time (Sec.)		
(Scenarios)	Method	Iter.	Total	Master	Objective
`sslp_5_25` (50)	Cutting plane	46	172	0.09	-121.6
	ℓ_∞ trust region	21	71	0.05	-121.6
	`ConicBundle`	15	55	0.38	-121.6
	OOQP	9	29	0.06	-121.6
	PIPS-IPM	9	29	0.10	-121.6
`sslp_10_50` (50)	Cutting plane	73	2508	0.90	-364.64
	ℓ_∞ trust region	71	4352	0.90	-364.64
	`ConicBundle`	27	1521	4.82	-364.64
	OOQP	14	571	0.18	-365.62
	PIPS-IPM	22	1004	0.44	-364.41
`sslp_15_45` (10)	Cutting plane	89	5088	0.20	-260.5
	ℓ_∞ trust region	65	5762	0.11	-260.5
	`ConicBundle`	36	1628	0.14	-260.5
	OOQP	39	2374	0.48	-260.5
	PIPS-IPM	38	2408	0.26	-260.5
`dcap233` (200)	Cutting plane	194	1189	116	1837.87
	ℓ_∞ trust region	58	295	46	1833.37
	`ConicBundle`	34	*7200	7027	*
	OOQP	58	337	69	1833.4
	PIPS-IPM	68	341	34	1833.4
`dcap243` (200)	Cutting plane	252	1920	168	2326.13
	ℓ_∞ trust region	40	189	25	2322.37
	`ConicBundle`	34	*7200	7048	*
	OOQP	68	348	106	2321.21
	PIPS-IPM	68	266	32	2321.21
`dcap332` (200)	Cutting plane	321	1348	189	1059.10
	ℓ_∞ trust region	47	210	74	1059.08
	`ConicBundle`	33	*7200	7079	*
	OOQP	77	363	121	1059.08
	PIPS-IPM	79	282	43	1059.10

linear algebra). Both OOQP and PIPS-IPM use the same algorithmic implementation of Mehrotra's predictor-corrector scheme [24], and each instance is solved from scratch. All mixed-integer subproblems are solved by using the software package SCIP [25].

Results are presented in Table 2. For each instance and solution method, we report the total number of iterations, the total execution time, the time spent solving the master program, and the objective value at convergence. First observe that the cutting-plane method requires the largest number of iterations, as expected. For the `sslp` instances, the proximal bundle methods are superior (in terms of total execution time) to the ℓ_∞ trust-region approach, while the trust-region approach appears to be superior on the `dcap` instances. Note the disagreement on objective values, in particular for the `dcap` instances. This is explained partially by differing convergence criteria, although in some cases numerical instability is also present; for example, the cutting-plane method for `dcap233` reports a mathematically invalid objective value that is larger than the optimal objective value of the original stochastic integer problem.

For the `sslp` instances, our implementation of the proximal bundle method has a comparable iteration count to that of the `ConicBundle` package, empirically confirming our algorithmic implementation. However, `ConicBundle` is unable to solve the `dcap` instances to completion because of the time spent solving the QP master. `ConicBundle` uses a similar interior-point method to that of our implementation; the difference in execution time is attributable to its use of dense linear algebra. That is, `ConicBundle` treats the Hessian matrix of the QP master as entirely dense, whereas in Section 4 it was shown to be highly structured. (Note that `ConicBundle` does not solve the equality-constrained formulation; the Hessian, however, remains highly structured.) The results demonstrate the computational advantage of using general sparse linear algebra routines within the interior-point method for solving (15). In addition, specialized linear algebra for the block-angular structure (as used within PIPS-IPM) may produce further improvements even before considering parallel computation.

5.2. *Parallel experiments*

A preliminary parallel version of the proximal bundle method (Figure 2) was implemented by using the Message Passing Interface (MPI) API. In our implementation, each scenario is statically assigned to an *MPI process*, which may be considered a parallel worker. This worker is then responsible for solving the mixed-integer subproblems for its assigned scenarios at each iteration. This static assignment is simpler to implement but is expected to be inferior in its load-balancing properties compared to dynamically assigning the subproblems.

Table 3 contains results from parallel experiments with instances similar to those of the serial experiments in Table 2 but with larger numbers of scenarios. Unlike in Table 2, we consider only the proximal bundle method. Again we solve the master QP using the general sparsity-exploiting QP solver OOQP and then using the structure-exploiting solver PIPS-IPM. PIPS-IPM is run in parallel using the same MPI processes as the mixed-integer subproblems. Identical runs were performed with 1, 8, 16, and 32 parallel processes, each corresponding to a physical processing core. Recall that the nodes of the compute cluster have 8 cores each; hence 32 processes corresponds to 4 physical nodes.

There is a range of behavior on the five instances displayed. On all instances, using PIPS-IPM results in significant speedups in the time to solve the master QP as the number of processes increases, although with diminishing marginal improvements. For the `dcap` instances, solving the master QP forms a significant portion of the total serial execution time. Therefore, by solving the master in parallel, in addition to the mixed-integer subproblems, significant reductions in the total execution time are observed. In comparison, the execution time of the `sslp` instances is dominated by the mixed-integer subproblems, and so speedups in the master have a smaller effect.

Perhaps the most surprising result is that the speedups in solving the mixed-integer subproblems are smaller than expected. For example, `sslp_10_50` exhibits much less than 2x speedup from 8 to 16 processes. These results can be explained by the high variability in the time to solve the subproblems as well as by the lack of dynamic load balancing in our implementation. Although the behavior can be explained in retro-

Table 3: Parallel experiments with the proximal bundle method. With OOQP, only the MIP subproblems are solved in parallel; with PIPS-IPM, both the MIP subproblems and the master QP are solved in parallel. For the `dcap` instances, significant overall speedups are observed as a result of reducing the time spent solving the QP master.

		Serial Sparse QP solver (OOQP)				Parallel QP Solver (PIPS-IPM)			
Instance	Parallel		Time/Iter. (Sec.)				Time/Iter. (Sec.)		
(Scenarios)	Processes	# Iter.	Total	Master	Objective	# Iter.	Total	Master	Objective
`sslp_5_25` (100)	1	8	6.31	0.013	-127.370	8	6.33	0.022	-127.370
	8	8	1.14	0.011	-127.370	8	1.21	0.064	-127.370
	16	8	0.71	0.013	-127.370	8	0.74	0.007	-127.370
	32	8	0.54	0.013	-127.370	8	0.53	0.007	-127.370
`sslp_10_50` (500)	1	26	3,301	0.44	-349.132	22	2,939	0.24	-349.133
	8	31	1,252	0.50	-349.132	24	1,049	0.08	-349.136
	16	27	1,224	0.43	-349.131	28	1,005	0.04	-349.136
	32	31	1,106	0.53	-349.137	27	865	0.05	-349.118
`dcap233` (500)	1	68	16.15	4.53	1,736.678	66	12.71	1.29	1,736.674
	8	68	6.62	4.37	1,736.678	70	2.39	0.20	1,736.681
	16	68	5.75	4.38	1,736.678	73	1.56	0.13	1,736.681
	32	68	9.91	4.35	1,736.678	70	1.24	0.12	1,736.674
`dcap243` (500)	1	57	14.37	3.05	2,165.479	57	12.11	0.98	2,165.479
	8	57	5.04	2.97	2,165.479	58	2.12	0.15	2,165.492
	16	57	4.00	2.97	2,165.479	59	2.07	0.09	2,165.490
	32	57	7.26	2.95	2,165.479	59	1.88	0.10	2,165.495
`dcap332` (500)	1	82	13.51	5.04	1,587.435	80	9.45	1.59	1,587.256
	8	82	6.65	4.96	1,587.435	79	1.70	0.20	1,587.391
	16	82	5.81	4.98	1,587.435	80	1.89	0.12	1,587.123
	32	82	11.20	4.95	1,587.435	77	1.43	0.11	1,587.439
`dcap342` (500)	1	59	14.78	2.76	1,902.842	71	12.07	1.26	1,903.014
	8	59	6.03	2.70	1,902.842	67	3.19	0.16	1,903.214
	16	59	5.46	2.71	1,902.842	56	2.77	0.08	1,902.893
	32	59	8.05	2.70	1,902.842	62	2.60	0.08	1,902.894

spect, the solution of the subproblems is typically expected to be a "trivially" parallel computation, yet here it is seen to be far from such. Further work will be required to address this issue, perhaps by considering asynchronicity akin to the work of [12].

Returning to the scalability of the master QP, we conducted an experiment evaluating the relative performances of OOQP and PIPS-IPM on instances with larger numbers of first-stage variables. As noted in §3, formulation (15) may not be efficient unless $K \times r >> n_1$, where n_1 is the number of first-stage variables. Both the `sslp` and `dcap` instances have a small number of first-stage variables (Table 1). Because we are not aware of SMIP instances with a larger number of first-stage variables, we use the linear `prod` instances with 1,000 scenarios (generated by simple Monte Carlo sampling). This substitution is valid because the structure of the QP master remains the same, and we do not consider the time spent in the (now linear) subproblems. The results in Figure 3 demonstrate that as the number of first-stage variables increases, the general sparse QP solver may become more effective than the structure-exploiting solver in serial; yet, when run in parallel, the structure-exploiting solver is significantly faster.

Acknowledgments

This work was supported in part by the U.S. Department of Energy under Contract DE-AC02-06CH11357. This research used resources of the Laboratory Computing Resource Center at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Financial support from the University of Chicago Booth School of Business is also gratefully acknowledged. We thank Christoph Helmberg for correspondence regarding the `ConicBundle` package.

References

[1] J. R. Birge, F. Louveaux, Introduction to Stochastic Programming, 2nd Edition, Springer, New York, 2011.

[2] S. Sen, Algorithms for stochastic mixed-integer programming models, in: K. Aardal, G. L. Nemhauser, R. Weismantel (Eds.), Discrete Optimization, Elsevier, 2005, pp. 515–558.

[3] C. C. Carøe, R. Schultz, Dual decomposition in stochastic integer programming, Operations Research Letters 24 (1-2) (1999) 37–45.

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

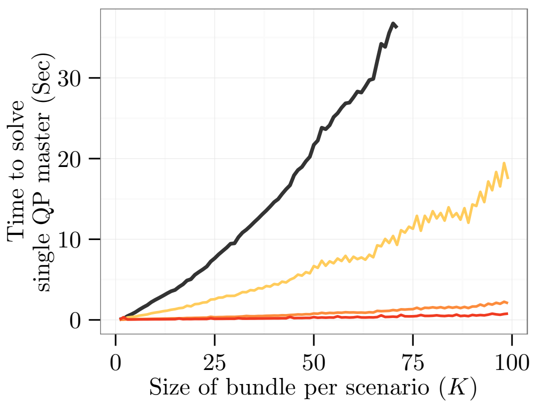
[5] A. Frangioni, About Lagrangian methods in integer optimization, Annals of Operations Research 139 (1) (2005) 163–193.

[6] C. Lemaréchal, Lagrangian relaxation, in: M. Jünger, D. Naddef (Eds.), Computational Combinatorial Optimization, Springer, 2001, pp. 112–156.

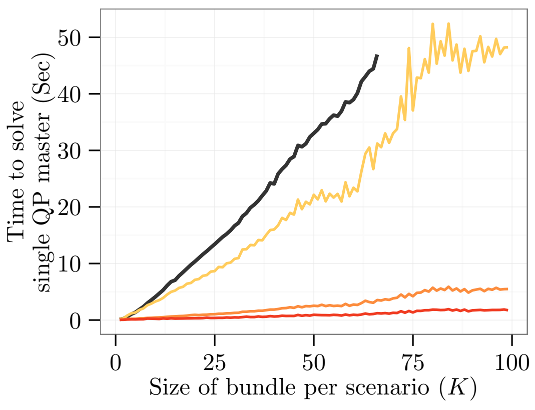
[7] J. B. Hiriart-Urruty, C. Lemaréchal, Convex Analysis and Minimization Algorithms, Vol. I-II, Springer-Verlag, Germany, 1993.

[8] K. C. Kiwiel, Approximations in proximal bundle methods and decomposition of convex programs, Journal of Optimization Theory and Applications 84 (3) (1995) 529–548.

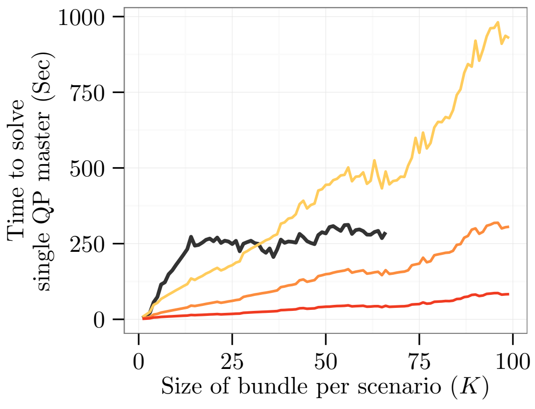
[9] W. Römisch, S. Vigerske, Recent progress in two-stage mixed-integer stochastic programming with applications to power production planning, in: P. M. Pardalos, S. Rebennack, M. V. F. Pereira, N. A. Iliadis, P. M. Pardalos (Eds.), Handbook of Power Systems, Vol. I, Springer, 2010, pp. 177–208.



(a) n_1 = 50 first-stage variables



(b) n_1 = 250 first-stage variables



(c) n_1 = 1,500 first-stage variables

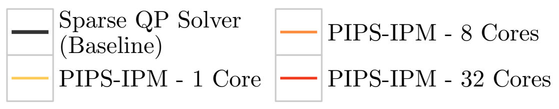


Figure 3: Time to solve QP master for small, medium, and large `prod` instances, with $r = 1,000$ scenarios each. Black line terminates when the sparse QP solver (OOQP) failed due to out-of-memory error. PIPS-IPM is a parallel block-angular-structure-exploiting QP solver applied to (15). Number of first-stage variables and size of the bundle (number of columns) per scenario primarily determine the difficulty of solving the QP. Parallel speedups at the largest bundle size for (1 to 8 cores, 1 to 32 cores) are (8.5x, 22x), (8.8x, 28x), and (3.0x, 11x), for the small, medium, and large instances, respectively.

[10] A. Ruszczyński, A regularized decomposition method for minimizing a sum of polyhedral functions, Mathematical Programming 35 (3) (1986) 309–333.

[11] C. Lemaréchal, A. Nemirovskii, Y. Nesterov, New variants of bundle methods, Mathematical Programming 69 (1) (1995) 111–147.

[12] J. Linderoth, S. Wright, Decomposition algorithms for stochastic programming on a computational grid, Computational Optimization and Applications 24 (2) (2003) 207–250.

[13] C. I. Fábián, Z. Szöke, Solving two-stage stochastic programming problems with level decomposition, Computational Management Science 4 (4) (2007) 313–353.

[14] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, F. Vanderbeck, Comparison of bundle and classical column generation, Mathematical Programming 113 (2) (2008) 299–344.

[15] V. Zverovich, C. Fábián, E. Ellison, G. Mitra, A computational study of a solver system for processing two-stage stochastic LPs with enhanced benders decomposition, Mathematical Programming Computation 4 (3) (2012) 211–238.

[16] G. M. Amdahl, Validity of the single processor approach to achieving large scale computing capabilities, in: Proceedings of AFIPS spring joint computer conference, ACM, Atlantic City, NJ, 1967, pp. 483–485.

[17] J. Gondzio, A. Grothey, Parallel interior-point solver for structured quadratic programs: Application to financial planning problems, Annals of Operations Research 152 (1) (2007) 319–339.

[18] K. C. Kiwiel, A Cholesky dual method for proximal piecewise linear programming, Numerische Mathematik 68 (3) (1994) 325–340.

[19] J. Forrest, CLP, available at `https://projects.coin-or.org/Clp`. Accessed December 27, 2012.

[20] C. Helmberg, ConicBundle, available at `http://www-user.tu-chemnitz.de/~helmberg/ConicBundle/`. Accessed December 27, 2012.

[21] E. M. Gertz, S. J. Wright, Object-oriented software for quadratic programming, ACM Transactions on Mathematical Software 29 (1) (2003) 58–81.

[22] I. S. Duff, MA57: A code for the solution of sparse symmetric definite and indefinite systems, ACM Transactions on Mathematical Software 30 (2) (2004) 118–144.

[23] M. Lubin, C. G. Petra, M. Anitescu, V. Zavala, Scalable stochastic optimization of complex energy systems, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, Seattle, WA, 2011, pp. 64:1–64:10.

[24] S. Mehrotra, On the implementation of a primal-dual interior point method, SIAM Journal on Optimization 2 (4) (1992) 575–601.

[25] T. Achterberg, SCIP: Solving constraint integer programs, Mathematical Programming Computation 1 (1) (2009) 1–41.