Branching in the Dual Decomposition Method for Stochastic Integer Programs*

Zhichao Ma[1]
and Jeff Linderoth[1,2]

[1] Department of Industrial and Systems Engineering
[2] Wisconsin Institute of Discovery
{zma59,linderoth}@wisc.edu

Abstract. Stochastic mixed-integer programs are hard to solve directly. Instances with more than a modest number of scenarios cannot be solved by passing their deterministic equivalent (extensive form) to a mixed-integer programming solver. By employing Dual Decomposition, which relaxes the nonanticipativity constraints in a Lagrangian fashion, the problem can be divided into smaller, independent subproblems. Then, there is a search over the space of Lagrange multipliers to make the lower bound obtained from the relaxation as large as possible. However, even if the optimal solution of the Dual Decomposition is found, there is no guarantee of solving the original problem—the nonconvexity induced by integrality can result in a gap between the optimal solution and the lower bound provided by the Lagrangian Dual. To close this gap, we must enforce consensus among the variables that violate the nonanticipativity constraints by branching. Usually there are many variables that do not agree, and thus many choices for the branching entity.
In this work, we study branching for the Dual Decomposition method in stochastic integer programming. We translate and implement traditional methods from mixed integer programming to this context, including max fractionality, pseudocosts, and strong branching. We also present a hybrid method, that uses the Lagrangian weighted deviation as a selection filter for strong branching. We present an empirical study comparing the performance of all methods on a variety of instances. We find that the new hybrid method typically outperforms other methods.

Keywords: Stochastic Integer Programming · Branching Methods · Dual Decomposition

1 Introduction

We study Lagrangian dual decomposition methods for solving the following two-stage stochastic mixed integer program:

$$\phi^{\text{SMIP}} = \min_{x,y} \left\{ c^\top x + \sum_{s \in S} p_s q^{s\top} y^s : (x, y^s) \in K^s,\ s \in S \right\}, \tag{SMIP}$$

* Work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-0

where $K_s := \{(x, y^s) : W^s y^s = h^s - T^s x,\ x \in X,\ y^s \in Y\}$, and X and Y are mixed-integer sets defined by linear inequalities and integrality restrictions on the variables x and y, respectively. The elements $(q^s, h^s, T^s, W^s) := \xi^s$ are typically samples of a random vector $\boldsymbol{\xi}$, occurring with probability p_s. In (SMIP) the decisions x are made before $\boldsymbol{\xi}$ is observed, and the objective is to optimize the cost of x plus the expected cost of recourse decisions y^s that are made after observing sample ξ^s corresponding to scenario $s \in S$.

To form the Lagrangian dual, we introduce copies of the first-stage decision variables for every scenario along with a "master copy" z, yielding an equivalent formulation of (SMIP):

$$\phi^{\text{SMIP}} = \min_{x,y,z} \Big\{ \sum_{s\in S} p_s(c^\top x^s + {q^s}^\top y^s) : (x^s, y^s) \in K^s,\ x^s = z,\ s \in S \Big\}. \quad (1)$$

The constraints $x^s = z,\ s \in S$, enforce *nonanticipativity*, that is, the same first-stage decisions must be made for all scenarios. Introducing multiplier vectors $\lambda^1, \lambda^2, \ldots, \lambda^{|S|}$ for the nonanticipativity constraints (NAC), we obtain the Lagrangian dual function for (1):

$$\mathcal{L}(\lambda) := \min_{x,y,z} \Big\{ \sum_{s\in S} p_s(c^\top x^s + {q^s}^\top y^s + {\lambda^s}^\top (x^s - z)) : (x^s, y^s) \in K^s,\ s \in S \Big\}. \quad (2)$$

For any choice of the vectors of multipliers $\lambda := (\lambda^1, \lambda^2, \ldots, \lambda^{|S|})$, The Lagrangian function $\mathcal{L}(\lambda)$ provides a lower bound on ϕ^{SMIP}. Moreover, evaluation of the function $\mathcal{L}(\lambda)$ decomposes by scenario and thus can be implemented in a distributed manner. After its introduction by Carøe and Schultz [8], many authors have successfully applied the Lagrangian dual decomposition technique to obtain high-quality solutions to (SMIP) [3, 16, 15, 21, 26].

However, even if the optimal choice of multipliers λ^* yielding the largest lower bound $\mathcal{L}(\lambda^*)$ is found, the lack of convexity of the sets K_s can result in an optimality gap: $\mathcal{L}(\lambda^*) - \phi^{\text{SMIP}} > 0$. To close the gap, the (relaxed) nonanticipativity constraints must be enforced. After evaluating $\mathcal{L}(\lambda)$, there are typically many variables whose values do not agree, and thus a branching decision must be made to decide on which variable(s) to enforce consensus first. In this paper, we study branching methods that enforce consensus among the first-stage variables in the Lagrangian split-variable formulation (1).

Branching methods have been well-studied for mixed integer programming (MIP) [1, 2, 6, 20, 7, 23, 12], and we will draw inspiration and intuition from MIP branching methods. A common (yet ineffective) method for branching in MIP is most infeasible branching, which chooses the variable whose fractional part is closest to 0.5 [1]. Another simple branching method in MIP is selecting the fractional variable the largest objective function coefficient [24]. More complex MIP branching methods estimate or compute the improvement in objective function value after branching, as in pseudocost branching [11], strong branching [5], and reliability branching [1].

The choice of branching entity can have a *huge* impact on the size of the enumeration tree in MIP [1, 10, 20], so it is surprising that besides the recent

work [17] we are unaware of any work that has investigated techniques for making branching decisions in the context of dual decomposition for stochastic programs.

The remainder of the paper is organized into 3 sections. In Section 2, we gives some details of our implementation of the dual decomposition method. Section 3 describes five different methods for selecting the branching variable in the dual decomposition method. In Section 4, we present computational results comparing the empirical performance of the branching methods.

2 Dual Bounds

2.1 Computing a Lagrangian Bound

The *Lagrangian dual* is to find the λ that maximizes $\mathcal{L}(\lambda)$ defined in (2). Note that since z is unrestricted, the function $\mathcal{L}(\lambda)$ is bounded below only when $\sum_{s\in S} \lambda^s = 0$. When this requirement holds, the z variables vanish from (2), and the evaluation of $\mathcal{L}(\lambda)$ becomes separable over the different scenarios. We can write

$$\mathcal{L}(\lambda) = \sum_{s\in S} p_s \mathcal{L}_s(\lambda^s), \tag{3}$$

where

$$\mathcal{L}_s(\lambda^s) := \min_{x,y}\{c^\top x + q^{s\top} y + \lambda^{s\top} x : (x,y) \in K^s\} \quad \text{for all } s \in S, \tag{4}$$

and the Lagrangian dual can be written

$$\phi^{\mathrm{LD}} := \max_{\lambda}\Big\{\mathcal{L}(\lambda) : \sum_{s\in S} \lambda^s = 0\Big\}. \tag{LD}$$

The functions $\mathcal{L}_s(\lambda^s)$ are piecewise-concave functions of λ^s for each $s \in S$, so $\mathcal{L}(\lambda)$ is a piecewise-concave function of $\mathcal{L}(\lambda)$. Many methods exist for solving this non-differential concave maximization problem, such as the classical subgradient method [27], cutting-plane methods [14], column-generation methods (applied to the dual) [22], level methods [18], and bundle methods [16] to name but a few.

In our implementation, we used a simple (asychronous) subgradient method designed for SMIP to find every improving values of $\mathcal{L}(\lambda)$ [19]. Given values of multipliers at the kth iteration of the algorithm $\lambda^k := (\lambda^{k,1}, \lambda^{k,2}, \ldots \lambda^{k,s})$, evaluating the Lagrangian function $\mathcal{L}(\lambda^k)$ requires solving the optimization problem (4) for each $s \in S$. We assume for simplicity that each optimization problem has an optimal solution $x^{k,s}$ for each $s \in S$. Using these optimal solution values, the values of λ^k are updated according to

$$\lambda^{k+1,s} = \lambda^{k,s} + a^k(x^{k,s} - \sum_{s'=1}^{|S|} p_s x^{k,s'}), \tag{5}$$

where a_k is the well-known Polyak stepsize [25]

$$a^k = \frac{\mathcal{L}^* - \mathcal{L}(\lambda^k)}{\sum_{s=1}^{|S|} \|x^{k,s} - \sum_{s'=1}^{|S|} p_s x^{k,s'}\|^2},$$

and ϕ^{LD} is the optimal Lagrangian function value. In practice, an upper bound on the optimal function value is used in place of the unknown ϕ^{LD}.

We could have used a more sophisticated method for optimizing (LD), but our focus is on evaluating the empirical performance of branching methods and their ability to improve the dual bound obtained at a node of the branch and bound tree, so this simple multiplier-update rule should suffice. Moreover, in practice, the problem (LD) is not solved to optimality. Rather, multipliers are updated for perhaps a fixed number of iterations. To save computational effort and improve empirical performance, we often do a modest number of subgradient iterations at each node of the branch and bound tree.

2.2 Branching in Dual Decomposition

As stated in the introduction, the nonconvexity of the sets K^s can result in a gap between the optimal value ϕ^{LD} of the Lagrangian dual and the optimal solution value ϕ^{SMIP}. If there is a gap, there will be at least one variable x_i that violates the NAC. That is, the values of $x_i^1, ..., x_i^{|S|}$ in the optimal solutions to the scenario subproblems are not all equal. We can attempt to enforce consensus by partitioning the domain at a branching point. In this work, similar to [8], we use the average value $\bar{x}_i$ as the branching point

$$\bar{x}_i := \sum_{s=1}^{|S|} p_s x_i^s. \tag{6}$$

In [17], Kim and Dandurand consider a different branch-point selection and provide finite termination guarantees. We also implemented different branch point selection mechanisms, but they did not significantly alter the performance of the branching methods compared to the simple midpoint scheme.

At node N of the branch and bound tree, we define the scenario subproblem s as $K^{s,N}$, and our branching dichotomy will create two new branching nodes L and R. If variable x_i is required to take on integer values, then the feasible regions of the children are

$$K^{s,L} := K^{s,N} \cap \{(x,y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} : x_i \leq \lfloor \bar{x}_i \rfloor\} \; \forall s \in S \tag{7}$$

$$K^{s,R} := K^{s,N} \cap \{(x,y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} : x_i \geq \lceil \bar{x}_i \rceil\} \; \forall s \in S. \tag{8}$$

If x_i is a continuous variable that violates the NAC, the feasible regions of the children are

$$K^{s,L} := K^{s,N} \cap \{(x,y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} : x_i \leq \bar{x}_i - \epsilon\} \; \forall s \in S \tag{9}$$

$$K^{s,R} := K^{s,N} \cap \{(x,y) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} : x_i \geq \bar{x}_i + \epsilon\} \; \forall s \in S, \tag{10}$$

for some small feasibility tolerance $\epsilon > 0$.

It's important to note that for some of the scenario subproblems, the solution of its relaxation x_i^s will not violate the new branching constraint $x_i \leq \bar{x}_i$ or $x_i \geq \bar{x}_i$. In our implementation, we reuse the solution for those scenarios when evaluating the Lagrangian at the node.

3 Branching Methods

If there are multiple variables that violate the NAC at a node, then we must select one on which to branch. Good branching variables are those that significantly improve the lower bounds of their child nodes. In this section, we briefly review branching methods from MIP and explain their adaption and implementation in the context of dual decomposition for solving (SMIP).

For instances in which both continuous and integer variables violate NAC at a node, we use a *branching priority* to branch on all integer variables first. That is, if there are integer variables violating the NAC, continuous variables violating the NAC are not candidates for branching. Because the branching constraints (7) and (8) allow for rounding of the new bound, these are somewhat stronger branches, and our preliminary computational experiments clearly demonstrated that setting this branching priority improved computational performance.

3.1 Most Infeasible = Max Dispersion

The most-fractional branching rule in MIP selects the branching variable whose value most violates the (relaxed) integrality conditions. Similarly, the Max Dispersion branching rule in DD selects the branching variable whose values $x_i^1, x_i^2, \ldots x_i^{|S|}$ most violate the (relaxed) NAC. Here, we define *most* as having the largest variance, so the Max Dispersion branching rules selects the index i^*_{MD} with

$$i^*_{MD} \in \underset{i\in\{1,\ldots,n_1\}}{\operatorname{argmax}} \sum_{s\in S} (x_i^s - \bar{x}_i)^2. \tag{11}$$

3.2 Lagrangian-Weighted Max Deviation

It is well-known in MIP that the most-fractional branching rule is empirical terrible, essentially no-better than selecting a random variable on which to branch [1]. The rule completely ignores the objective function when making a decision. Intuitively, changing the bounds for variables with large objective function coefficients should have more impact on the optimal solution value than for variables with small coefficients. In the Lagrangian (3), the objective function coefficient for variable x_i depends both on c_i and the λ_i^s for each scenario $s \in S$. The Lagrangian-Weighted Max Deviation branching method selects the branching variable i^*_{LWMD} with

$$i^*_{LWMD} \in \underset{i\in\{1,\ldots,n_1\}}{\operatorname{argmax}} \sum_{s\in S} p_s(c_i + \lambda_i^s)^2 (x_i^s - \bar{x}_i)^2. \tag{12}$$

We use the square of the objective function coefficients in order to to match the (squared) units of deviation from the mean.

3.3 Strong Branching

Strong branching is an important component of most variable selection rules used in commercial MIP solvers. Strong branching works by explicitly calculating bounds of potential child nodes and selecting the branching variable that leads to the largest bound improvement in the children. It is straightforward to adopt this methodology for branching in the DD method. If ℓ_N is the lower bound obtained at node N of the branch and bound tree, and $I_N \subset [n_1]$ are the indices of variables that do not satisfy the NAC, then for each $i \in I_N$, the two branching nodes (from (7) and (8) or from (9) and (10)) are created, and a fixed number s of subgradient iterations are done for these tentative branches. Note that in the context of the DD method, doing one subgradient iteration requires solving an integer program for every scenario, so iterations requires significant computational effort.

We denote ℓ_i^L and ℓ_i^R as the largest lower bound in each of these nodes during the subgradient iterations and define the bound improvement parameters

$$\Delta\ell_i^L = \ell_i^L - \ell_N \tag{13}$$

$$\Delta\ell_i^R = \ell_i^R - \ell. \tag{14}$$

The Strong Branching variable selection rule branches on the variable index i_{SB}^* with

$$i_{SB}^* = \underset{i\in\{1,\ldots,n_1\}}{\operatorname{argmax}} \ \texttt{score}(\Delta\ell_i^L, \Delta\ell_i^R), \tag{15}$$

where

$$\texttt{score}(a,b) = (1-\mu)\min\{a,b\} + \mu\max\{a,b\} \tag{16}$$

is a score function designed to combine the bound improvement measure of both child nodes recommended by [1, 20]. We used a value of $\mu = 0.1$ in our implementation.

During strong branching on variable x_i, we can potentially add bounds to the node based on the observed lower bound value. Specifically, if the observed lower bound is larger than the value of a known upper bound, we can fix the bounds to those on the sibling branch. If both sibling branches have lower bounds larger than a known upper bound, the node can be fathomed.

3.4 Pseudocost and Reliability Branching

Pseudocosts are values that attempt to estimate the per-unit change on dual bound obtained when branching on a specific variable. Specifically, we associate with each variable $i \in [n_1]$ two quantities Ψ_i^L and Ψ_i^R that are estimates of the rate of change of the dual bound if we reduce variable x_i's upper bound or increase variable x_i's lower bound, respectively. Given solutions $x_i^1, x_i^2, \ldots x_i^{|S|}$

at a node, with branching point $\bar{x}_i$, we can compute the weighted total bound changes for both child potential child nodes as

$$F_i^L := \sum_{s \in S} p_s \max\{x_i^s - \bar{x}_i, 0\}$$
$$F_i^R := \sum_{s \in S} p_s \max\{\bar{x}_i - x_i^s, 0\}.$$

The Pseudocost Branching variable selection method selects the variable index with the largest score

$$i^*_{PSEUDO} \in \underset{i \in \{1,\ldots,n_1\}}{\operatorname{argmax}} \ \texttt{score}(\Psi_i^L F_i^L, \Psi_i^R F_i^R). \tag{17}$$

It remains to define exactly how the pseudocost values Ψ_i^L and Ψ_i^R are computed. After branching on variable x_i and observing the change in bound of each child node, we obtain information useful for estimating pseudocosts. Specifically, after branching, observing dual bound changes on the left and right branches of $\Delta\ell_i^L$ and $\Delta\ell_i^R$, we can compute an *implied pseudocost* as

$$\Psi_i^L = \sum_{s \in S : x_i^s > \bar{x}} p_s \Delta\ell_i^L / (x_i^s - \bar{x}) \tag{18}$$

$$\Psi_i^R = \sum_{s \in S : x_i^s < \bar{x}} p_s \Delta\ell_i^R / (\bar{x} - x_i^s) \tag{19}$$

Then, we can use the average observed value of the implied pseudocosts for variable x_i as its pseudocosts. An important factor not yet considered is how the pseudocost values are initialized.

Reliability branching is an implementation of pseudocost branching that does a careful initialization of the pseudocost values for each variable [1]. Specifically, the pseudocosts Ψ_i^L and Ψ_i^R for variable x_i are considered *unreliable* if variable x_i has not been branched on at least η_{rel} times. When considering branching on a variable, if the variable is unreliable, its bound changes are computed via strong branching as described in Section 3.3 equations (13) and (14), and these values are used when making the branching decision. If we define $U \subset [n_1]$ as the set of unreliable variables and $[n_1] \setminus U$ as the set of reliable variables, then the Reliability Branching variable selection method choose the variable with index i^*_{REL}, where

$$i^*_{REL} \in \arg \max_{i \in U, j \in [n_1] \setminus U} \{\texttt{score}(\Delta\ell_i^L, \Delta\ell_i^R), \texttt{score}(\Psi_j^L F_j^L, \Psi_j^R F_j^R)\}. \tag{20}$$

Moreover, the bound changes computed via strong-branching give rise to an implied pseudocost (18) which can be averaged into the current pseudocost estimates Ψ_i^L and Ψ_i^R thus making variable x_i somewhat more "reliable."

3.5 Filter Branching Method

Computational results with strong branching demonstrate that it is the most effective method at reducing the number of nodes of a branch and bound tree,

but computing the estimates (13) and (14) requires significant computational effort. A final proposed branching method "filters" the branching candidates by selecting the top K variables when ranked according to the weighted Lagrangian metric in (12), and performs then takes strong branching on these K candidate variables. Finally, we branch on the candidate variable with the highest score from strong branching.

4 Computational Results

4.1 Implementation Details

The DD method was implemented in C++ using SUTIL [9] to read and manipulate the stochastic programming instances, Cplex for solving the MIP scenario subproblems, and the runtime framework MW [13] for managing the algorithm coordination between master problem and subproblems. MW is designed to help implement master-worker-type parallel algorithms running in a dynamic, heterogeneous, distributed computing environment. For the computations comparing DD branching rules, MW was used in its *synchronous* mode, so all computations were run on a single machine for each instance.

Computational experiments were run on a cluster of shared computing resources at the UW-Madison Center for High Throughput Computing managed by the HTCondor scheduling framework [28]. To ensure a consistent comparison of computational results run on different machines at different times, we use the *number of mixed integer programs solved* as the primary measure of the units of work, and we limit all methods to an upper limit of $6000|S|$ MIPs solved. Our experiments are designed to compare how well different branching rules improve the Lagrangian lower bound. To reduce variability introduced by the branch and bound process, we set the upper-bound cutoff to the value of the best-known solution to each problem instance.

Other parameters of the algorithm were set as follows:

- The subgradient method was initialized with first iterate $\lambda = 0$ and run for at most 500 iterations at the root node.
- At each child node, the subgradient method was initialized with the λ that had largest Lagrangian value from its parent and run for at most either 1 or 5 iterations.
- The nodes of the branch and bound tree were processed in best-bound order (smallest active node lower bound first).
- A feasibility tolerance of $\epsilon = 10^{-6}$ was used when branching on continuous variables in (9) and (10).
- We use $s = 1$ subgradient iterations to estimate the lower bound improvements ℓ_i^L and ℓ_i^R in strong branching. (Section 3.3)
- We use $\eta_{\text{rel}} = 2$ to define whether or not a variable's pseudocost is reliable. (Section 3.4)
- In the filter branching method of Section 3.5, we consider at most $K = 5$ candidates for strong branching.

4.2 Problem Instances

Computational experiments comparing the different branching methods were run on five families of problem instances. The `sizes` and `dcap` instance families are from SIPLIB [4], and three other instances (`sssslp`, `facility` and `facility-binary`) are new instance families we generated. Each member of the instance family may have a different number of scenarios, resulting in 31 instances in our benchmark set. Table 1 shows characteristics of all instances.

Table 1: Detailed characteristics of problem instances

		First Stage				Second Stage			
Name	Scenario	Var	Int	Bin	Row	Var	Int	Bin	Row
sizes	3,5,10	75	0	10	37	75	0	10	37
dcap233	200,300,500	12	6	0	6	27	27	0	15
dcap243	200,300,500	12	6	0	6	36	36	0	18
dcap332	200,300,500	12	6	0	6	24	24	0	12
f3_h6_c3	10,20,50	36	36	0	37	66	57	9	54
f5_h5_c5	10,20,50	50	50	0	51	155	130	25	90
f5_h7_c10	10,50	105	105	0	106	410	360	50	180
f5_h8_c10_binary	10,20,50	240	0	240	120	410	0	0	135
f10_h15_c20_binary	10,20	900	0	900	450	3020	0	0	480
sssslp_10_20	10,20,50	10	0	10	0	420	220	200	260
sssslp_15_30	10,20,50	15	0	15	0	930	480	450	540

4.3 Comparison of Branching Methods

We ran our implementation of the DD algorithm on each of the 31 instances using each of the five branching methods described in Section 3. There were 15 of the 31 instances solved to optimality by at least one of the branching methods and 16 unsolved by any of the methods within the work limit of $6000|S|$ MIPs. It is notable that *none* of the instances in the sizes or dcap family were solved to optimality within the work limit, and these are the instance families in our test suite that have continuous first-stage decision variables. We use different metrics for comparing the methods for the different sets of instances. For solved instances, we use the number of MIPs solved, and for unsolved instances, we use the final remaining optimality gap. Recall that for all instances, we set the upper bound value to the best-known solution value, so we are really comparing the lower bound achieved. We ran the experiment with limits of both 1 and 5 subgradient iterations at each child node.

Figure 1 shows performance profiles summarizing the results of all experiments. The left figure show results for the solved instances, using either 1 or 5 subgradient iterations per child node, and the right figures show results for unsolved instances, again using 1 or 5 subgradient iterations per node.

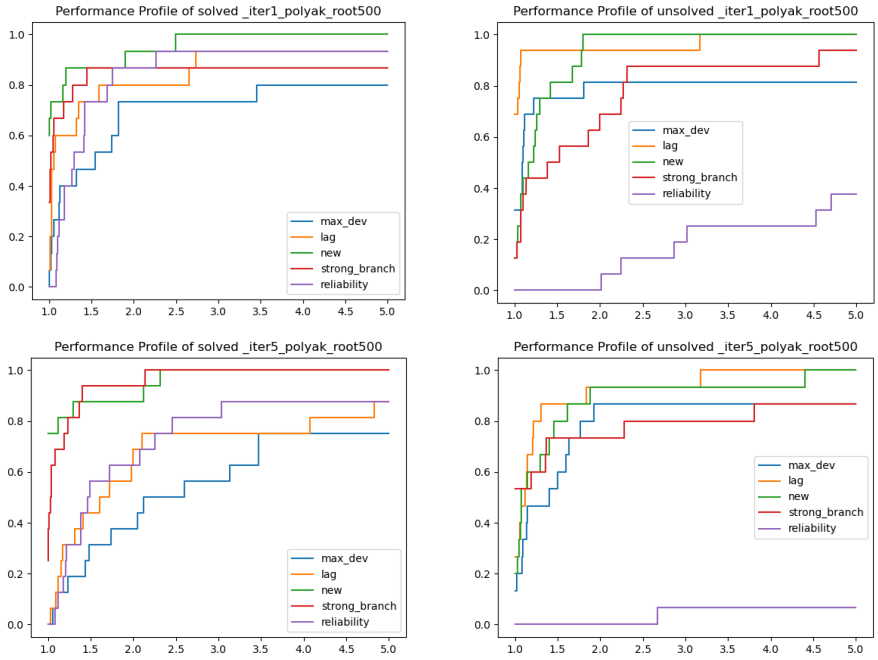

Fig. 1: Performance Profile of solved and unsolved problems instances with 1 and 5 iterations in child nodes

For solved instances, the strong branching method or the filtered strong branching method was most effective at reducing the total number of MIPs solved in order to prove optimality. For unsolved instances, max-weighted Lagrangian and the filter strong branching method are the most effective.

The pseudocost-based method of reliability branching and surprisingly poor performance, especially for the unsolved instances. Understanding why this is the case and adapting pseudocost-based methods for dual-decomposition remains an area of future research.

5 Conclusion

We empirically compared five different branching rules for closing the optimality gap in a Lagrangian dual decomposition method for solving two stage stochastic mixed integer programs. The results indicate that judicious use of strong-branching methodology in this context can outperform other branching rules considered. Continuing work includes examination of the empirical tradeoff between applying subgradient iterations to improve the dual bound at each node or using subgradient iterations to select a good branching candidate.

References

1. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33**, 42–54 (2004)
2. Achterberg, T., Berthold, T.: Hybrid branching. In: van Hoeve, W.J., Hooker, J.N. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 309–311. Springer Berlin Heidelberg (2009)
3. Ahmed, S.: A scenario decomposition algorithm for 0-1 stochastic programs. Operations Research Letters **41**, 565–569 (2013)
4. Ahmed, S., Garcia, R., Kong, N., Ntaimo, L., Parija, G., Qiu, F., Sen, S.: SIPLIB: a stochastic integer programming test problem library (2015), https://www2.isye.gatech.edu/ sahmed/siplib
5. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem. Princeton University Press, Princeton, NJ (2006)
6. Berthold, T., Salvagnin, D.: Cloud branching. In: Gomes, C., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 28–43. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
7. Breu, R., Burdet, C.A.: Branch and bound experiments in zero-one programming. Mathematical Programming **2**, 1–50 (1974)
8. Carøe, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Operations Research Letters **24**, 37–45 (1999)
9. Czyzyk, J., Linderoth, J., Shen, J.: Sutil: A utility library for handling stochastic programs (2005), https://coral.ise.lehigh.edu/ sutil/index.html
10. Forrest, J.J.H., Hirst, J.P.H., Tomlin, J.A.: Practical solution of large scale mixed integer programming problems with UMPIRE. Management Science **20**, 736–773 (1974)
11. Gauthier, J.M., Ribière, G.: Experiments in mixed-integer linear programming using pseudo-costs. Mathematical Programming **12**(1), 26–47 (1977)
12. Glankwamdee, W., Linderoth, J.T.: Lookahead branching for mixed integer programming. In: Proceedings of the Twelfth INFORMS Computing Society Meeting. pp. 130–150 (2011)
13. Goux, J.P., Kulkarni, S., Linderoth, J., Yoder, M.: An enabling framework for master-worker applications on the computational grid. In: Proceedings the Ninth International Symposium on High-Performance Distributed Computing. pp. 43–50 (2000)
14. Kelley, J.E.: The cutting plane method for solving convex programs. Journal of SIAM **8**(4), 703–712 (1960)
15. Kim, K., Petra, C.G., Zavala, V.M.: An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming. SIAM Journal on Optimization **29**(1), 318–342 (2019)
16. Kim, K., Zavala, V.M.: Algorithmic innovations and software for the dual decomposition method applied to stochastic mixed-integer programs. Mathematical Programming Computation **10**(2), 225–266 (2018)
17. Kim, K., Dandurand, B.: Scalable branching on dual decomposition of stochastic mixed-integer programming problems. Mathematical Programming Computation **14**, 1–41 (2022)
18. Lemaréchal, C., Nemirovskii, A., Nesterov, Y.: New variants of bundle methods. Mathematical Programming **69**, 111–147 (1995)

19. Lim, C.H., Linderoth, J.T., Luedtke, J.R., Wright, S.J.: Parallelizing subgradient methods for the lagrangian dual in stochastic mixed-integer programming. INFORMS Journal on Optimization **3**(1), 1–22 (2021)
20. Linderoth, J.T., Savelsbergh, M.W.P.: A computational study of search strategies in mixed integer programming. INFORMS Journal on Computing **11**, 173–187 (1999)
21. Lubin, M., Martin, K., Petra, C.G., Sandikci, B.: On parallelizing dual decomposition in stochastic integer programming. Operations Research Letters **41**(3), 252 – 258 (2013)
22. Lulli, G., Sen, S.: A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. Management Science **50**(6), 786–796 (2004)
23. Mitra, G.: Investigation of some branch and bound strategies for the solution of mixed integer linear programs. Mathematical Programming **4**, 155–170 (1973)
24. Padberg, M.W., Rinaldi, G.: A branch and cut algorithm for the solution of large scale traveling salesman problems. SIAM Review **33**, 60–100 (1991)
25. Polyak, B.: Introduction to Optimization. Optimization Software (1987)
26. Ryan, K., Rajan, D., Ahmed, S.: Scenario decomposition for 0-1 stochastic programs: Improvements and asynchronous implementation. In: Proceedings of the Parallel and Distributed Processing Symposium Workshops. pp. 722–629. IEEE (2016)
27. Shor, N.: Minimization methods for non-differentiable functions. Springer-Verlag (1985)
28. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The Condor experience. Concurrency and Computation: Practice and Experience (2005)