On Adaptive Multicut Aggregation for Two-Stage Stochastic Linear Programs with Recourse

Svyatoslav Trukhanov and Lewis Ntaimo[1]
Department of Industrial and Systems Engineering, Texas A&M University, 3131 TAMU, College Station, TX 77843, USA, slavik@tamu.edu and ntaimo@tamu.edu

Andrew Schaefer
Department of Industrial Engineering, 1048 Benedum Hall, University of Pittsburgh, Pittsburgh, PA 15261, schaefer@ie.pitt.edu

Abstract

Outer linearization methods for two-stage stochastic linear programs with recourse, such as the L-shaped algorithm, generally apply a single optimality cut on the nonlinear objective at each major iteration, while the multicut version of the algorithm allows for several cuts to be placed at once. In general, the L-shaped algorithm tends to have more major iterations than the multicut algorithm. However, the tradeoffs in terms of computational time is problem dependent. This paper investigates the computational trade-offs of adjusting the level of optimality cut aggregation from single cut to pure multicut. Specifically, an adaptive multicut algorithm that dynamically adjusts the aggregation level of the optimality cuts in the master program, is presented and tested on standard large scale instances from the literature. The computational results reveal that a cut aggregation level that is between the single cut and the multicut results in computational times of up to over 50% reduction compared to the single cut method.

Keywords: stochastic programming, multicut aggregation, adaptive cuts

1 Introduction

Outer linearization methods such as the L-shaped algorithm (Van Slyke and Wets, 1969) for stochastic linear programming (SLP) problems with recourse generally apply a single cut on the nonlinear objective at each major iteration, while the multicut version of the algorithm (Birge, 1988) allows for cuts up to the number of outcomes (scenarios) to be placed at once. In general, the L-shaped algorithm tends to have more major iterations than the multicut algorithm. However, the trade-offs in terms of computational time may be problem dependent. This paper generalizes these two approaches through an adaptive multicut algorithm that dynamically adjusts the level of aggregation of the optimality cuts in the master program during the course of the algorithm. Results of our computational investigation suggest that for certain classes of problems the adaptive multicut provides better performance than the traditional single cut and multicut methods.

The relative advantages of the single cut and multicut methods have already been explored (Birge, 1988, Gassmann, 1990). Citing earlier work (Birge, 1988, Gassmann,

[1]Corresponding Author

1990), Birge and Louveaux (1997) gave the rule of thumb that the multicut method is preferable when the number of scenarios is not much larger than the size of the first stage decision dimension space. In earlier work, Birge (1985) used variable aggregation to reduce the problem size and to find upper and lower bounds on the optimal solution. Ruszczynski (1988) considered adding a regularized term to the objective function to stabilize the master problem and reported computational results showing that the method runs efficiently despite solving a quadratic problem instead of a linear one. The number of iterations that required extensive computations decreased twice for the instances considered.

More recent work related to modified versions of the L-shaped method examined serial and asynchronous versions of the L-shaped method and a trust-region method (Linderoth and Wright, 2003). Algorithms were implemented on a grid computing platform that allowed authors to solve large scale problem instances from the literature. These include the SSN (Sen et al., 1994) with 10^5 scenarios and STORM (Mulvey and Ruszczynski, 1992) with 10^7 scenarios. The authors reported CPU time improvements of up to 83.5% for instances of SSN with 10^4 scenarios.

The main contributions of this paper are twofold: 1) an adaptive optimality multicut algorithm for SLP, and 2) computational investigation with standard instances from the literature demonstrating that changing the level of optimality cut aggregation in the master problem during the course of the algorithm can lead to significant improvement in solution time. The results of this work also have potential to influence the design and implementation of future algorithms, especially those based on Benders decomposition (Benders, 1962). For example, decomposition methods based on the the sample average approximation method (Shapiro, 2000), the trust-region method (Linderoth and Wright, 2003) and the stochastic decomposition method (Higle and Sen, 1991), may benefit from the adaptive multicut approach.

The rest of this paper is organized as follows. The next section gives the problem statement and a motivation for the adaptive multicut approach. A formal description of the method is given in Section 3. Computational results are reported in Section 4. The paper ends with some concluding remarks in Section 5.

2 An Adaptive Multicut Approach

A two-stage stochastic linear program with fixed recourse in the extensive form can be given as follows:

$$\text{Min } c^\top x + \sum_{s \in S} p_s q_s^\top y_s \tag{1a}$$

$$\text{s.t. } Ax = b \tag{1b}$$

$$T_s x + W y_s \geq r_s, \ \forall s \in S \tag{1c}$$

$$x \geq 0, \ y_s \geq 0, \ \forall s \in S \tag{1d}$$

where $x \in \mathbb{R}^{n_1}$ is the vector of first stage decision variables with the corresponding cost vector $c \in \mathbb{R}^{n_1}$; $A \in \mathbb{R}^{m_1 \times n_1}$ is the first stage constraint matrix; and $b \in \mathbb{R}^{m_1}$ is the

first stage righthand-side vector. Additionally, S is the set of scenarios (outcomes), and for each scenario $s \in S$, $y_s \in \mathbb{R}^{n_2}$ is the vector of second stage decision variables; p_s is the probability of occurrence for scenario s; $q_s \in \mathbb{R}^{n_2}$ is the second stage cost vector; $r_s \in \mathbb{R}^{m_2}$ is the second stage righthand-side vector; $T_s \in \mathbb{R}^{m_2 \times n_1}$ is the technology matrix; and $W \in \mathbb{R}^{m_2 \times n_2}$ is the recourse matrix.

To set the ground for the adaptive multicut method, we first make some observations from a comparison of the single cut L-shaped method and the multicut method in the context of problem (1). In general, the single cut L-shaped method tend to have 'information loss' due to the aggregation of all the scenario dual information into one optimality cut in the master problem at each major iteration of the method. On the other hand, the multicut method uses all scenario dual information and places an optimality cut for each scenario in the master problem. Thus the L-shaped method generally requires more major iterations than the multicut method. However, on iteration index k the size of the master problem in the multicut method grows much faster and is $(m_1 + k|S|) \times (n_1 + |S|)$ in size in the worst case, as compared to $(m_1 + k) \times (n_1 + 1)$ in the L-shaped method. Consequently, the multicut method is expected to have increased solution time when $|S|$ is very large.

Our goal is to devise an algorithm that performs better than the single cut L-shaped and the multicut methods. The insights to our approach are to 1) use more information from subproblems, which assumes adding more than one cut at each major iteration; and 2) keep the size of master problem relatively small, which requires to limit the number of cuts added to master problem. These insights lead us to consider a multicut method with 'partial' cut aggregation. This means that such an algorithm requires partitioning the sample space S into D *aggregates* $\{S_d\}_{d=1}^{D}$ such that $S = S_1 \cup S_2 \cup \cdots S_D$ and $S_i \cap S_j = \emptyset \ \forall i \neq j$. Then at the initialization step of the algorithm, optimality cut variables θ_d are introduced, one for each aggregate. The optimality cuts are also generated one per aggregate in the form $(\beta_d^k)^\top x + \theta_d \geq \alpha_d^k$, where $(\beta_d^k)^\top = \sum_{s \in S_d} p_s (\pi_s^k)^\top T_s$ and $\alpha_d^k = \sum_{s \in S_d} p_s (\pi_s^k)^\top r_s$.

The multicut and the single cut L-shaped methods have two extremes of optimality cut aggregation. The multicut L-shaped method has *no* cut aggregation, which corresponds to the case when $D = |S|$ and $S_d = \{s_d\},\ d = 1, \cdots, |S|$. The expected recourse function value is approximated by having an optimality decision variable for each scenario in the master program. On the contrary, the single cut L-shaped method has the *highest* level of cut aggregation, which means $D = 1$ and $S_1 = S$. The expected recourse function value is approximated by only one optimality decision variable in the master program. According to our numerical results, the optimal runtime is achieved on some middle level of aggregation $1 < D < |S|$ but this level is not known a priori.

Our approach allows for the optimality cut aggregation to be done dynamically during the course of the algorithm. Thus the master program will have 'adaptive' optimality variables, that is, the number of optimality variables will change during the course of the algorithm. The goal is to let the algorithm learn more information about the expected recourse function and then settle for a level of aggregation that leads to faster

convergence to the optimal solution. Therefore, we propose initializing the algorithm with a 'low' level of cut aggregation and increasing it, if necessary, as more information about the expected recourse function is learned during the course of the algorithm. As the level of cut aggregation increases, the number of optimality variables in the master problem decreases. If $|S|$ is not 'very large' one could initialize the algorithm as pure multicut, where an optimality variable θ_s is created for each $s \in S$ in the master program. Otherwise, an initial level of cut aggregation between 1 and $|S|$ should be chosen. Computer speed and memory will certainly influence the level of cut aggregation since a low level of cut aggregation would generally require more computer memory.

3 The Basic Adaptive Multicut Algorithm

To formalize our approach, at the k-th iteration let the scenario set partition be $\mathcal{S}(k) = \{S_1, S_2, \cdots, S_{l_k}\}$, where the sets $S_i, i = 1, \cdots, l_k$ are disjoint, that is, $S_i \cap S_j = \emptyset \ \forall i \neq j$ and $\cup_{i=1}^{l_k} S_i = S$. Therefore, each scenario $s \in S$ belongs to only one $S_i \in \mathcal{S}(k)$ and the aggregate probability of S_i is defined as $p_{S_i} = \sum_{s \in S_i} p_s$ with $\sum_{i=1}^{l_k} p_{S_i} = \sum_{s \in S} p_s = 1$. Let $\mathcal{F}(k)$ denote the set of iteration numbers up to k where all subproblems are feasible and optimality cuts are generated. Then the master program at major iteration k takes the form:

$$\text{Min } c^\top x + \sum_{d \in \mathcal{S}(k)} \theta_d, \tag{2a}$$

$$\text{s.t. } Ax = b \tag{2b}$$

$$(\beta^t)^\top x \geq \alpha^t, \ t \in \{1, \cdots, k\} \setminus \mathcal{F}(k) \tag{2c}$$

$$(\beta_d^t)^\top x + \theta_d \geq \alpha_d^t, \ t \in \mathcal{F}(k), \ d \in \mathcal{S}(k) \tag{2d}$$

$$x \geq 0 \tag{2e}$$

where constraints (2c) and (2d) are the feasibility and optimality cuts, respectively. We are now in a position to formally state the adaptive multicut algorithm.

Basic Adaptive Multicut Algorithm

Step 0: ***Initialization.***
Set $k \leftarrow 0$, $\mathcal{F}(0) \leftarrow \emptyset$, and initialize $\mathcal{S}(0)$.

Step 1: ***Solve the Master Problem.***
Set $k \leftarrow k + 1$ and solve problem 2. Let $(x^k, \{\theta_d^k\}_{d \in \mathcal{S}(k)})$ be an optimal solution to problem (2). If no constraint (2d) is present for some $d \in \mathcal{S}(k)$, θ_d^k is set equal to $-\infty$ and is ignored in the computation.

Step 2: ***Update Cut Aggregation Level.***

Step 2a ***Cut Aggregation.***

Generate aggregate $\mathcal{S}(k)$ using $\mathcal{S}(k-1)$ based on some aggregation rules. Each element of $\mathcal{S}(k)$ is a union of some elements from $\mathcal{S}(k-1)$. If according to aggregation rules $d_1, d_2, \cdots, d_j \in \mathcal{S}(k-1)$ are aggregated into $d \in \mathcal{S}(k)$, then $d = \bigcup_{i=1}^{j} d_i$ and $p_d = \sum_{i=1}^{j} p_{d_i}$. Master problem (2) will be modified by removing variables $\theta_{d_1}, \cdots, \theta_{d_j}$ and introducing the new one θ_d.

Step 2b ***Update Optimality Cuts.***

Update the optimality cut coefficients in the master program (2) as follows. For each iteration that optimality cuts were added define

$$\alpha_d^t = p_d \sum_{l=1}^{j} (1/p_{d_l}) \alpha_{d_l}^t \ \forall d \in \mathcal{S}(k), t \in \mathcal{F}(k) \tag{3}$$

and

$$\beta_d^t = p_d \sum_{l=1}^{j} (1/p_{d_l}) \beta_{d_l}^t \ \forall d \in \mathcal{S}(k), t \in \mathcal{F}(k). \tag{4}$$

For all iterations $t \in \mathcal{F}(k)$ replace cuts corresponding to $d_1, \cdots, d_j$ with one new cut $(\beta_d^t)^\top x + \theta_d \geq \alpha_d^t$ corresponding to d.

Step 3: ***Solve Scenario Subproblems.***

For all $s \in S$ solve:

$$\text{Min } q_s^\top y \tag{5a}$$

$$\text{s.t. } Wy \geq r_s - T_s x^k, \tag{5b}$$

$$y \geq 0. \tag{5c}$$

Let π_s^k be the dual multipliers associated with an optimal solution of problem (5). For each $d \in \mathcal{S}(k)$ if

$$\theta_d^k < p_d \sum_{s \in d} \pi_s^k (r_s - T_s x^k) \tag{6}$$

define

$$\alpha_d^k = \sum_{s \in S_d} p_s (\pi_s^k)^\top r_s \tag{7}$$

and

$$(\beta_d^k)^\top = \sum_{s \in S_d} p_s (\pi_s^k)^\top T_s \tag{8}$$

If condition (6) does not hold for any $d \in \mathcal{S}(k)$, stop x^k is optimal. Otherwise put $\mathcal{F}(k) = \mathcal{F}(k-1) \cup \{k\}$ and go to Step 1.

If problem (5) is *infeasible* for some $s \in S$, let μ_k be the associated dual extreme ray and define

$$\alpha^k = \mu_k^\top r_s, \tag{9}$$

and

$$(\beta^k)^\top = \mu_k^\top T_s \tag{10}$$

Go to Step 1.

Note that in the last step of the algorithm, if some scenario subproblem is infeasible, another implementation would be to find all infeasible subproblems and generate feasibility cut from each one to add to the master program. Since convergence of the above algorithm follows directly from that of the multicut method, a formal proof of convergence is omitted. An issue that remains to be addressed is the *aggregation rule* that one can use to obtain $\mathcal{S}(k)$ from $\mathcal{S}(k-1)$. Here we simply provide a basic aggregation rule that we implemented in our computational study.

Aggregation Rule:

- *Redundancy Threshold δ.* This rule is based on the observation that inactive (redundant) optimality cuts in the master program contain 'little' information about the optimal solution and can therefore be aggregated without information loss. Consider some iteration k after solving the master problem for some aggregate $d \in S(k)$. Let f_d be the number of iterations when optimality cuts corresponding to d are redundant, that is, the corresponding slack variables are nonzero. Also let $0 < \delta < 1$ be a given threshold. Then all d such that $f_d/|\mathcal{F}(k)| > \delta$ are combined to form one aggregate. This means that the optimality cuts obtained from aggregate d are often redundant at least δ of the time. This criterion works fine if the number of iterations is large. Otherwise, it is necessary to impose a 'warm up' period during which no aggregation is made so as to let the algorithm 'learn' information about the expected recourse function.

- *Bound on the Number of Aggregates.* As a supplement to the *redundancy threshold*, a bound on the minimum number of aggregates $|\mathcal{S}(k)|$ can be imposed. This prevents aggregating all scenarios prematurely, leading to the single cut L-shaped method. Similarly, a bound on the maximum number of the aggregates can also be imposed to curtail the proliferation of huge numbers of cuts in the master program. These bounds have to be set a priori and kept constant throughout the algorithm run.

Another viable aggregation rule is to aggregate optimality cuts with 'similar' gradients or orientation. We implemented this rule but did not observe significant gains in computation time due to the additional computational burden required for checking the cut gradients. Also, we considered deletion of 'old' cuts from the master program, that is, cuts that were added many iterations earlier and remained redundant for some time.

This seemed to reduce memory requirements but did not speed up the master program solves. Another consideration is partitioning an aggregate into two or more aggregates or cut 'disaggregation'. The major drawback to doing this is that all cuts should be kept in memory (or at least written to a file) in order to partition any aggregate. Thus this may be a computer memory intensive activity requiring careful book-keeping of old cuts for each scenario subproblem.

4 Computational Results

We implemented the adaptive multicut algorithm and conducted several computational experiments to gain insight into the performance of the algorithm. We tested the algorithm on several large scale instances from the literature. The three well-known SLP instances with fixed recourse, 20TERM, SSN, and truncated version of STORM were used for the study. 20TERM models a motor freight carrier's operations Mak et al. (1999), while SSN is a telephone switching network expansion planning problem Sen et al. (1994). STORM is a two period freight scheduling problem Mulvey and Ruszczynski (1992). The problem characteristics of these instances are given in Table 1.

Table 1: Problem Characteristics.

Problem	First Stage		Second Stage		No. of	No. of
	Rows	Cols	Rows	Cols	Random Variables	Scenarios
20TERM	3	63	124	764	40	$\approx 1.09 \times 10^{12}$
SSN	1	89	175	706	86	$\approx 1.02 \times 10^{70}$
STORM	59	128	526	1259	8	$\approx 3.91 \times 10^{5}$

Due to extremely large number of scenarios, solving these instances with all the scenarios is not practical and a sampling approach was used. Only random subsets of scenarios of sizes 1000, 2000 and 3000 were considered. In addition, sample sizes of 5000 and 10000 were used for STORM since it had the lowest improvement in computation time. Five replications were made for each sample size with the samples independently drawn. Both the static and dynamic cut aggregation methods were applied to the instances and the minimum, maximum and average CPU time, and the number of major iterations for different levels of cut aggregation recorded. All the experiments were conducted on an Intel Pentium 4 3.2 GHz processor with 512MB of memory running ILOG CPLEX 9.0 LP solver ILOG (2003). The algorithms were implemented in C++ using Microsoft .NET Visual Studio.

We designed three computational experiments to investigate 1) adaptive cut aggregation with limit on the number of aggregates, 2) adaptive cut aggregation based on the number of redundant optimality cuts in the master program, and 3) static aggregation where the level of cut aggregation is given. In the static approach, the cut aggregation level was specified a priori and kept constant throughout the algorithm. This experiment was conducted for different cut aggregation levels to study the effect on computational time and provide a comparison for the adaptive multicut method.

For each experiment we report the computational results as plots and put all the numerical results in tables in the Appendix. In the tables the column 'Sample Size' indicates the total number scenarios; 'Num Aggs' gives the number of aggregates; 'CPU Time' is the computation time in seconds; 'Min Iters', 'Aver Iters' and 'Max Iters' are the minimum, average and maximum major iterations (i.e. the number of times the master problem is solved); '% CPU Reduction' is the percentage reduction in CPU time over the single cut L-shaped method.

4.1 Experiment 1: Adaptive Multicut Aggregation

This experiment aimed at studying the performance of the adaptive multicut approach where the level of aggregation is dynamically varied during computation. As pointed out earlier, with adaptive aggregation one has to set a limit on the size of the aggregate to avoid all cuts being aggregated into one aggregate prematurely. In our implementation we used a parameter *agg-max*, which specified the maximum number of atomic cuts that could be aggregated into one to prevent premature total aggregation (single cut L-shaped). We initiated the aggregation level at 1000 aggregates (this is pure multicut for instances with sample size 1000). We arbitrarily varied *agg-max* from 10 to 100 in increments of 10, and up to 1000 in increments of 100. For instances with 3000 scenarios, values of *agg-max* greater than 300 resulted in insufficient computer memory and are thus not shown in the plots. We arbitrarily fixed the *redundancy threshold* at $\delta = 0.9$ with a "warm up" period of 10 iterations for all the instances.

The computational results showing how CPU time depends on aggregate size are summarized in the plots in Figures 1, 2, and 3. The results for 20TERM and STORM instances show that the number of major iterations increases with *agg-max*. However, the best CPU time is achieved at some level of agg-max between 10 and 100. The best CPU time is better than that of the single cut and the pure multicut methods. In this case there is over 60% reduction in CPU time over the single cut L-shaped method for 20TERM and over 90% for SSN. The results for STORM, however, show no improvement in computation time over the single method. This can be attributed to 'flat' expected recourse objective function for STORM, implying that no significant information is learned by using a multicut approach. In this case the single cut L-shaped method works better than both the adaptive multicut and the pure multicut methods.

4.2 Experiment 2: Adaptive multicut aggregation with Varying Threshold δ

In this experiment we varied the *redundancy threshold* δ in addition to varying *agg-max*. The parameter δ allows for aggregating cuts in the master program whose ratio of the number of iterations when the cut is not tight (redundant) to the total number of iterations since the cut was generated exceeds δ. The aim of this experiment was to study the performance of the method relative to the redundancy threshold δ. We arbitrarily chose values of δ between 0.5 and 1.0 with a fixed 'warm up' period of 10 iterations.

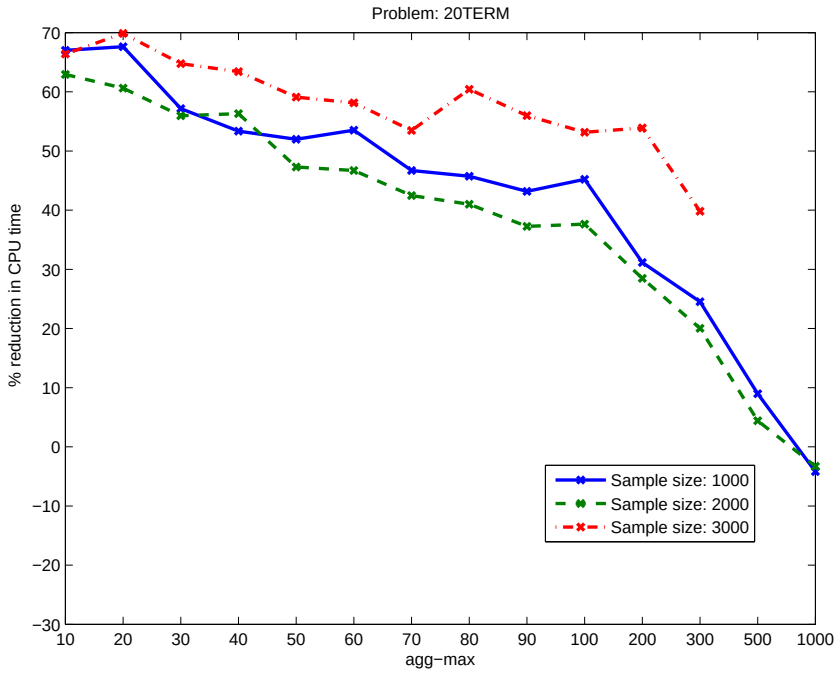

Figure 1: Adaptive multicut aggregation % CPU improvement over regular L-shaped method for 20TERM

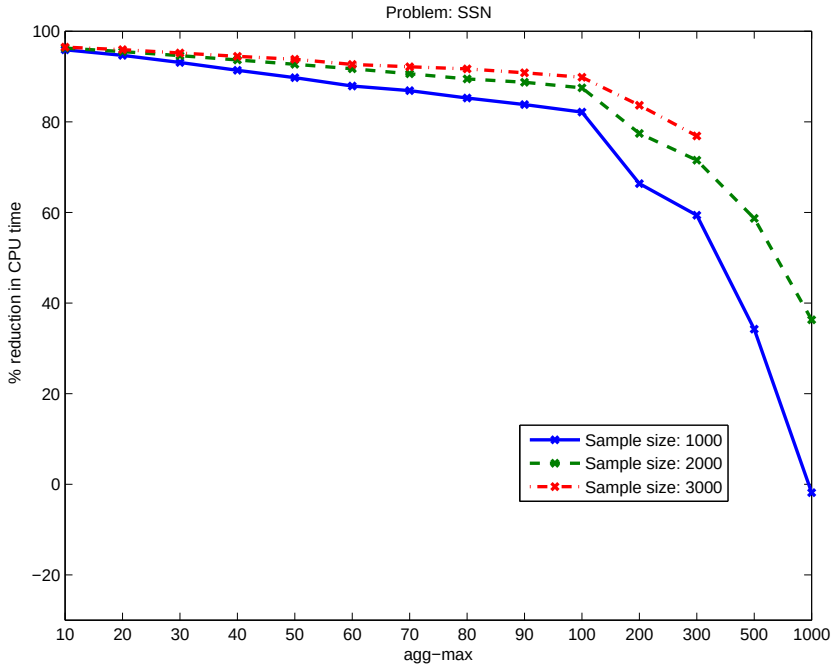

Figure 2: Adaptive multicut aggregation % CPU improvement over regular L-shaped method for SSN

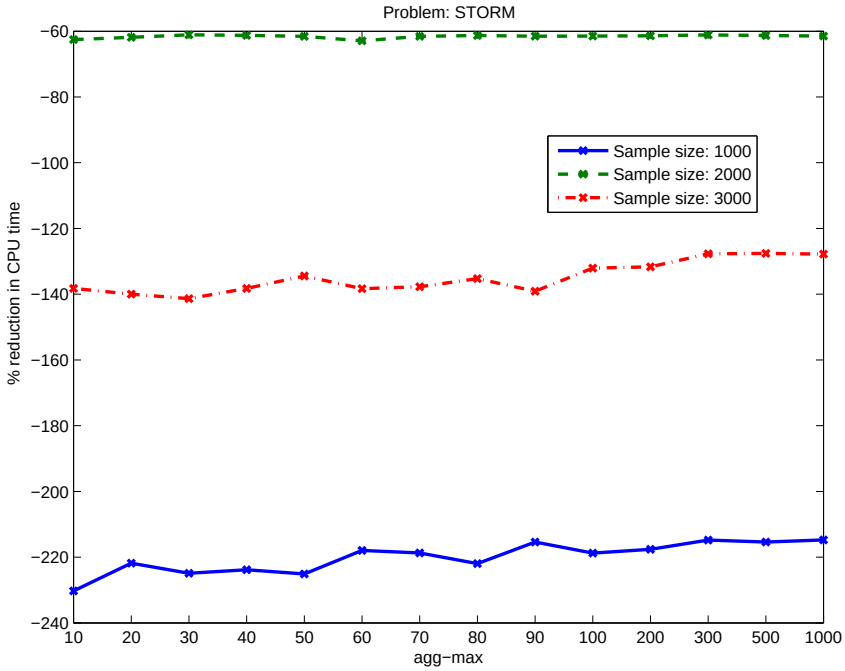

Figure 3: Adaptive multicut aggregation % CPU improvement over regular L-shaped method for STORM

Figures 4, 5 and 6 show results for values of delta 0.5, 0.6 and 0.7. The results seem to indicate that redundancy threshold is not an important factor in determining the CPU time for the adaptive multicut method.

4.3 Experiment 3: Static Multicut Aggregation

The aim of the third experiment was to study the performance of static partial cut aggregation relative to the single cut and the pure multicut methods. We arbitrarily set the level of aggregation a priori and fixed it throughout the algorithm. Thus the sample space was partitioned during the initialization step of the algorithm and the number of cuts added to master problem at each iteration step remained fixed. We varied the aggregation level from 'no aggregation' to 'total aggregation'. Under 'no aggregation' the number of aggregates is equal to the sample size, which is the pure multicut method. We only performed pure multicut for sample size 1000 due to the requirement of large amount of memory for the larger samples. 'Total aggregation' corresponds to the single cut L-shaped method. All static cut aggregations were arbitrarily made in a round-robin manner, that is, for a fixed number of aggregates m, cuts $1, m+1, 2m+1, \cdots$, were aggregated to aggregate 1, cuts $2, m+2, 2m+2, \cdots$, were aggregated to aggregate 2, and so on.

The computational results are summarized in the plots in Figures 7, 8, and 9. In the plots '% reduction in CPU time' is relative to the single cut L-shaped method, which acted as the benchmark. The results in Figures 7 show that a level of aggregation between single

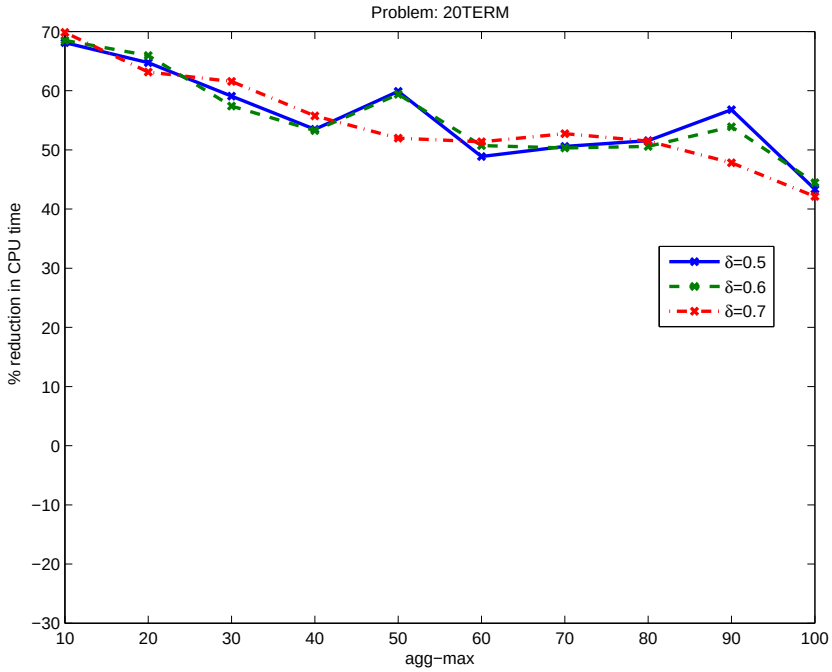

Figure 4: Adaptive multicut aggregation with δ % CPU improvement over regular L-shaped method for 20TERM

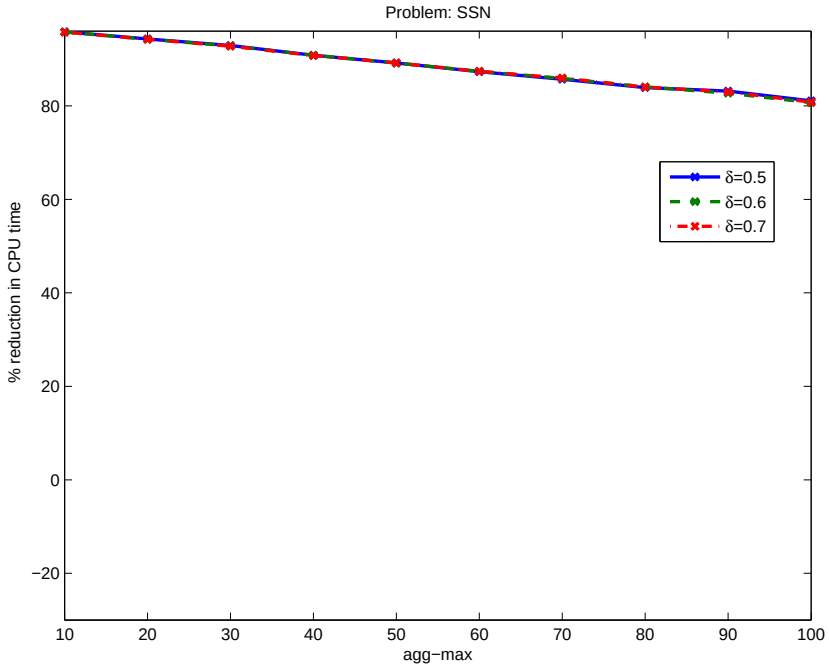

Figure 5: Adaptive multicut aggregation with δ % CPU improvement over regular L-shaped method for SSN

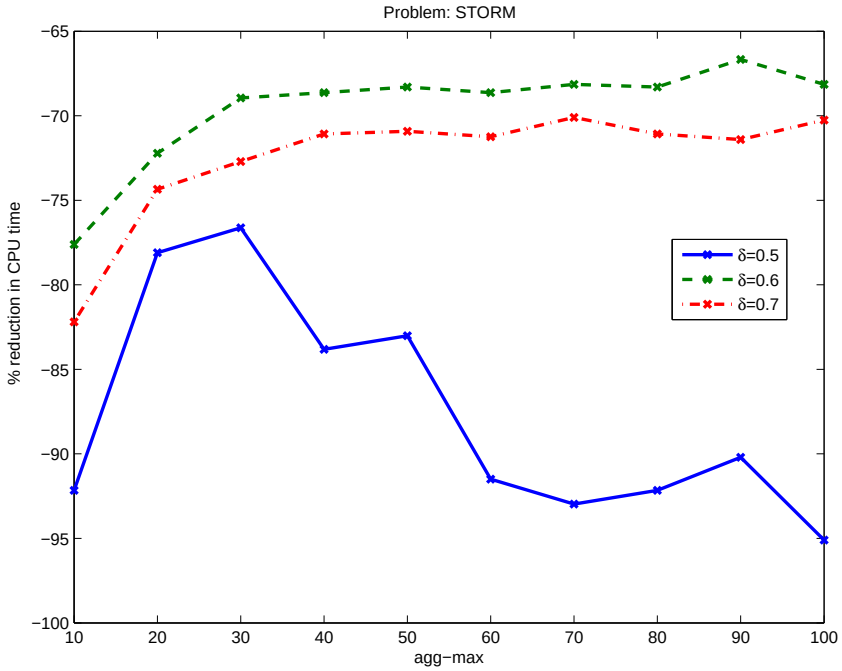

Figure 6: Adaptive multicut aggregation with δ % CPU improvement over regular L-shaped method for STORM

cut and pure multicut for 20TERM, results in about 60% reduction in CPU time over the L-shaped for all the instances. For SSN (Figure 8) percent improvements over 90% are achieved, while for STORM (Figure 9) no significant improvement (just over 10%) is observed. The results also show that increasing the number of aggregates decreases the number of iterations, but increases the size and time required to solve the problem. These results amply demonstrate that using an aggregation level that is somewhere between the single cut and the pure multicut L-shaped methods results in better CPU time. From a practical point of view, however, a good level of cut aggregation is not known a priori. Therefore, the adaptive multicut method provides a viable approach.

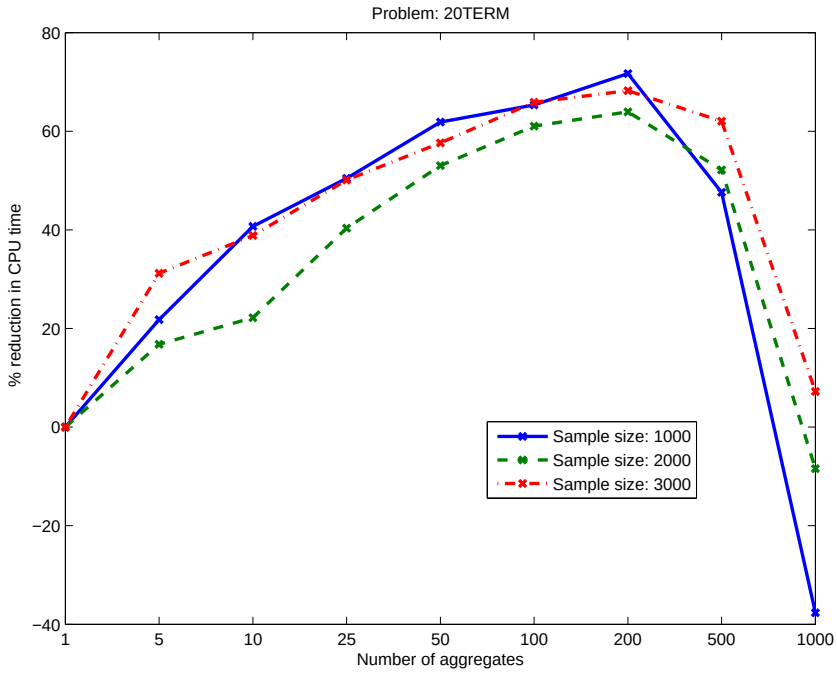

Figure 7: Static multicut aggregation % CPU improvement over regular L-shaped method for 20TERM

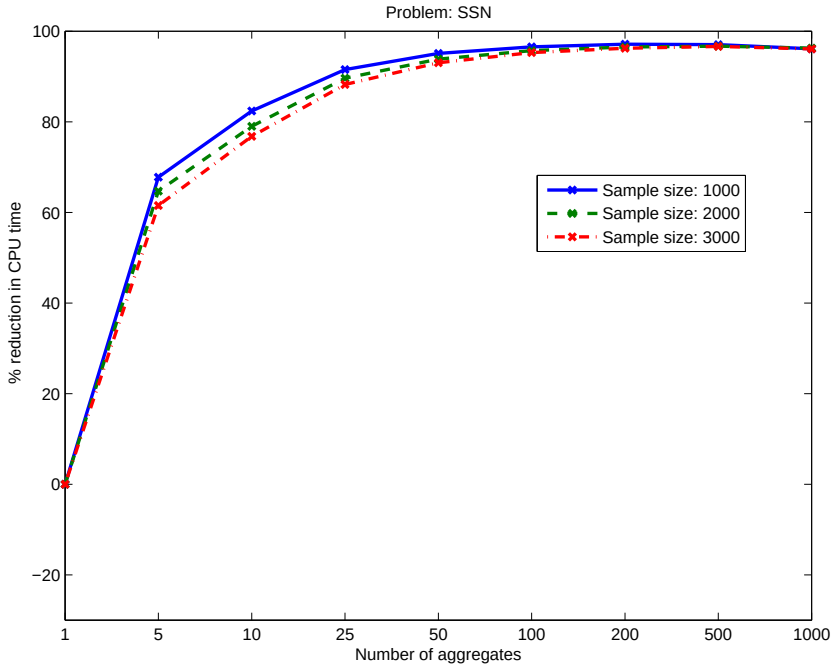

Figure 8: Static multicut aggregation % CPU improvement over regular L-shaped method for SSN

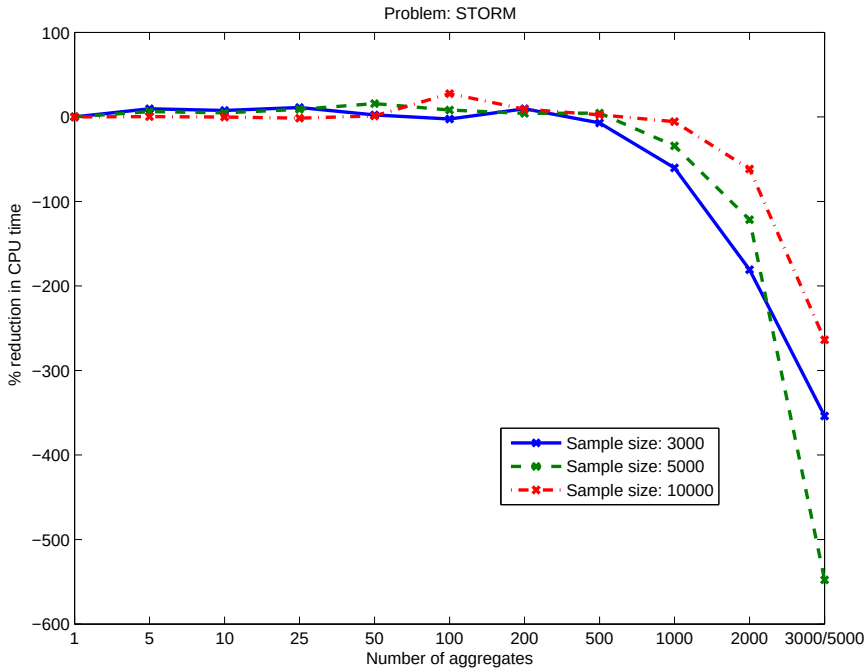

Figure 9: Static multicut aggregation % CPU improvement over regular L-shaped method for STORM

5 Conclusion

Traditionally, the single cut and multicut L-shaped methods are the algorithms of choice for stochastic linear programs. In this paper, we introduce an adaptive multicut method that generalizes the single cut and multicut methods. The method dynamically adjusts the aggregation level of the optimality cuts in the master program. A generic framework for building different implementations with different aggregation rules was developed and implemented. The computational results based on large scale instances from the literature reveal that the proposed method performs significantly faster than standard techniques. The adaptive multicut method has potential to be applied to decomposition-based stochastic linear programming algorithms. These include, for example, scalable sampling methods such as the sample average approximation method (Shapiro, 2000), the trust-region method (Linderoth and Wright, 2003), and the stochastic decomposition method (Higle and Sen, 1991, 1996, Higle and Zhao, 2007).

Acknowledgments. This research was supported by grants CNS-0540000, CMII-0355433 and CMII-0620780 from the National Science Foundation.

Appendix: Numerical Results

Table 2: Adaptive multicut aggregation for 20TERM

Sample Size	agg-max	CPU sec	Min Iters	Aver Iters	Max Iters	% CPU Reduction
1000	10	403.02	281	301.8	316	67.00
1000	20	395.43	311	359.8	396	67.62
1000	30	523.26	397	460.8	491	57.15
1000	40	569.79	499	521.2	533	53.34
1000	50	586.45	499	545.4	581	51.98
1000	60	567.58	482	551.4	603	53.52
1000	70	651.05	567	616.0	710	46.69
1000	80	662.77	585	631.2	664	45.73
1000	90	694.13	632	658.2	693	43.16
1000	100	669.08	528	658.4	757	45.21
1000	200	840.73	775	852.4	914	31.15
1000	300	921.68	890	914.2	954	24.53
1000	500	1111.70	1071	1139.6	1267	8.96
1000	1000	1272.22	983	1282.6	1491	-4.18
2000	10	803.99	262	285.6	300	62.93
2000	20	854.15	307	365.8	393	60.62
2000	30	955.18	409	429.8	461	55.96
2000	40	947.60	407	449.0	483	56.31
2000	50	1143.19	512	520.0	535	47.29
2000	60	1156.10	525	544.6	560	46.70
2000	70	1247.86	546	583.4	652	42.47
2000	80	1279.83	501	604.4	669	40.99
2000	90	1360.74	589	638.4	668	37.26
2000	100	1352.84	615	639.8	674	37.63
2000	200	1551.35	671	773.0	888	28.47
2000	300	1734.43	761	852.8	947	20.03
2000	500	2073.32	949	1014.2	1067	4.41
2000	1000	2240.14	1159	1224.6	1298	-3.28
3000	10	1350.50	244	259.8	281	66.39
3000	20	1211.21	305	327.2	362	69.86
3000	30	1415.92	391	404.8	423	64.76
3000	40	1469.87	400	443.2	472	63.42
3000	50	1643.93	446	476.8	504	59.09
3000	60	1682.43	448	496.0	523	58.13
3000	70	1869.92	533	556.4	574	53.47
3000	80	1590.16	480	513.8	561	60.43
3000	90	1768.80	511	556.4	591	55.98
3000	100	1881.62	530	592.4	626	53.17
3000	200	1852.93	600	632.6	667	53.89
3000	300	2418.09	836	845.4	856	39.82

Table 3: Adaptive multicut aggregation for SSN

Sample Size	agg-max	CPU sec	Min Iters	Aver Iters	Max Iters	% CPU Reduction
1000	10	81.23	60	61.8	64	95.88
1000	20	104.93	93	96.4	104	94.67
1000	30	135.92	131	135.8	139	93.10
1000	40	169.87	169	177.6	189	91.37
1000	50	202.05	212	218.6	229	89.74
1000	60	238.27	257	260.6	267	87.90
1000	70	258.44	276	287.8	303	86.88
1000	80	290.35	313	328.6	347	85.26
1000	90	318.80	341	362.6	376	83.81
1000	100	351.89	375	408.8	447	82.13
1000	200	662.72	751	789.2	851	66.35
1000	300	800.38	892	965.0	1021	59.36
1000	500	1295.09	1554	1646.4	1749	34.24
1000	1000	2006.02	2611	2806.8	3062	-1.85
2000	10	150.05	54	55.4	57	96.25
2000	20	180.33	80	81.6	83	95.50
2000	30	216.47	101	106.2	113	94.59
2000	40	254.05	129	131.8	133	93.65
2000	50	292.41	150	156.4	160	92.70
2000	60	332.53	173	181.6	193	91.70
2000	70	376.76	199	209.4	221	90.59
2000	80	422.19	232	236.8	241	89.46
2000	90	451.73	250	257.0	267	88.72
2000	100	501.13	276	288.6	314	87.48
2000	200	903.76	514	533.0	548	77.43
2000	300	1140.26	685	714.8	736	71.52
2000	500	1653.71	1064	1111.2	1176	58.70
2000	1000	2551.07	1714	1772.6	1854	36.29
3000	10	214.87	48	49.6	51	96.48
3000	20	249.74	70	72.0	73	95.91
3000	30	294.30	93	95.8	98	95.18
3000	40	339.46	112	116.0	119	94.44
3000	50	377.86	129	132.8	142	93.81
3000	60	449.29	157	162.6	170	92.65
3000	70	479.50	170	176.6	182	92.15
3000	80	507.89	187	189.8	195	91.69
3000	90	561.22	207	213.0	217	90.81
3000	100	621.87	223	236.4	250	89.82
3000	200	998.69	383	398.0	424	83.65
3000	300	1412.60	558	613.4	650	76.88

Table 4: Adaptive multicut aggregation for STORM

Sample Size	agg-max	CPU sec	Min Iters	Aver Iters	Max Iters	% CPU Reduction
1000	10	21.62	18	18.4	19	-230.28
1000	20	21.07	18	18.4	19	-221.82
1000	30	21.27	18	18.4	19	-224.88
1000	40	21.20	18	18.4	19	-223.85
1000	50	21.28	18	18.4	19	-225.09
1000	60	20.82	18	18.4	19	-217.95
1000	70	20.87	18	18.4	19	-218.72
1000	80	21.08	18	18.4	19	-221.97
1000	90	20.65	18	18.4	19	-215.40
1000	100	20.87	18	18.4	19	-218.75
1000	200	20.79	18	18.4	19	-217.62
1000	300	20.61	18	18.4	19	-214.81
1000	500	20.65	18	18.4	19	-215.39
1000	1000	20.61	18	18.4	19	-214.75
2000	10	21.56	17	17.0	17	-62.53
2000	20	21.47	17	17.0	17	-61.85
2000	30	21.36	17	17.0	17	-61.08
2000	40	21.39	17	17.0	17	-61.26
2000	50	21.43	17	17.0	17	-61.57
2000	60	21.61	17	17.0	17	-62.91
2000	70	21.43	17	17.0	17	-61.57
2000	80	21.39	17	17.0	17	-61.29
2000	90	21.42	17	17.0	17	-61.50
2000	100	21.41	17	17.0	17	-61.45
2000	200	21.41	17	17.0	17	-61.40
2000	300	21.37	17	17.0	17	-61.14
2000	500	21.39	17	17.0	17	-61.31
2000	1000	21.41	17	17.0	17	-61.45
3000	10	48.23	19	19.8	20	-138.26
3000	20	48.57	19	19.8	20	-139.98
3000	30	48.86	19	19.8	20	-141.37
3000	40	48.22	19	19.8	20	-138.25
3000	50	47.46	19	19.8	20	-134.47
3000	60	48.24	19	19.8	20	-138.31
3000	70	48.12	19	19.8	20	-137.72
3000	80	47.62	19	19.8	20	-135.28
3000	90	48.40	20	20.0	20	-139.11
3000	100	46.97	19	19.8	20	-132.06
3000	200	46.90	20	20.0	20	-131.69
3000	300	46.09	20	20.0	20	-127.71
3000	500	46.06	20	20.0	20	-127.57
3000	1000	46.11	20	20.0	20	-127.82

Table 5: Adaptive multicut aggregation with redundancy threshold δ

Problem Name	agg -max	CPU Time (secs.) δ			Aver Iters δ			% CPU Reduction δ		
		0.5	0.6	0.7	0.5	0.6	0.7	0.5	0.6	0.7
20term	10	407.01	402.15	384.90	308.4	304.6	297.4	68.11	68.49	69.84
20term	20	450.09	434.86	470.50	386.6	380.2	398.8	64.74	65.93	63.14
20term	30	522.31	543.72	490.48	461.8	469.2	452.0	59.08	57.40	61.57
20term	40	593.87	596.54	564.78	538.8	539.8	515.8	53.47	53.26	55.75
20term	50	512.02	518.50	612.98	511.8	514.4	565.8	59.88	59.38	51.97
20term	60	652.42	629.03	620.88	606.6	591.0	584.2	48.88	50.72	51.36
20term	70	630.61	634.00	603.24	602.6	603.2	590.6	50.59	50.33	52.74
20term	80	618.35	630.86	619.25	614.8	622.2	608.2	51.55	50.57	51.48
20term	90	551.51	588.42	666.14	582.0	602.2	650.2	56.79	53.90	47.81
20term	100	723.23	708.96	738.51	706.8	694.4	702.6	43.34	44.45	42.14
ssn	10	78.09	78.63	78.75	59.8	60.4	134.4	95.83	95.80	95.79
ssn	20	106.49	107.55	107.27	98.6	99.4	128.0	94.31	94.25	94.27
ssn	30	132.61	134.48	135.52	132.4	134.4	189.8	92.91	92.81	92.76
ssn	40	171.78	171.77	172.50	180.2	180.4	197.0	90.82	90.82	90.78
ssn	50	202.26	202.95	202.36	218.6	218.8	206.8	89.19	89.16	89.19
ssn	60	237.44	235.90	235.76	262.2	259.4	223.4	87.31	87.39	87.40
ssn	70	267.75	264.22	262.49	296.4	293.8	225.0	85.69	85.88	85.97
ssn	80	301.38	298.50	298.74	339.0	336.2	236.2	83.90	84.05	84.04
ssn	90	315.24	323.30	318.39	360.2	366.8	266.4	83.16	82.72	82.99
ssn	100	354.43	361.52	358.93	412.4	420.2	402.8	81.06	80.68	80.82
storm	10	11.76	10.87	11.15	21.4	18.4	18.4	-92.16	-77.61	-82.19
storm	20	10.90	10.54	10.67	19.2	18.4	18.4	-78.10	-72.22	-74.35
storm	30	10.81	10.34	10.57	19.0	18.4	18.4	-76.63	-68.95	-72.71
storm	40	11.25	10.32	10.47	21.0	18.4	18.4	-83.82	-68.63	-71.08
storm	50	11.20	10.30	10.46	20.8	18.4	18.4	-83.01	-68.30	-70.92
storm	60	11.72	10.32	10.48	22.8	18.4	18.4	-91.50	-68.63	-71.24
storm	70	11.81	10.29	10.41	23.0	18.4	18.4	-92.97	-68.14	-70.10
storm	80	11.76	10.30	10.47	22.8	18.4	18.4	-92.16	-68.30	-71.08
storm	90	11.64	10.20	10.49	22.2	18.0	18.4	-90.20	-66.67	-71.41
storm	100	11.94	10.29	10.42	23.2	18.4	18.4	-95.10	-68.14	-70.26

Table 6: Static multicut aggregation for 20TERM

Sample Size	Num aggs	CPU sec	Min Iters	Aver Iters	Max Iters	% CPU Reduction
1000	1	1221.17	1161	1304.2	1400	0.00
1000	5	955.19	988	1003.0	1023	21.78
1000	10	724.08	631	734.4	789	40.71
1000	25	605.16	515	569.0	608	50.44
1000	50	465.67	385	411.4	431	61.87
1000	100	423.27	311	320.2	327	65.34
1000	200	345.65	191	221.4	244	71.69
1000	500	639.72	149	160.2	167	47.61
1000	1000	1681.18	127	138.8	146	-37.67
2000	1	2168.89	1082	1232.6	1480	0.00
2000	5	1804.45	830	976.2	1054	16.80
2000	10	1687.90	666	853.6	919	22.18
2000	25	1294.14	611	643.0	684	40.33
2000	50	1018.93	448	483.8	523	53.02
2000	100	844.92	350	372.8	388	61.04
2000	200	781.94	241	283.6	302	63.95
2000	500	1037.75	182	194.4	201	52.15
2000	1000	2351.56	139	154.0	166	-8.42
2000	2000	12065.45	114	126.8	136	-456.30
3000	1	4018.32	1184	1426.2	1672	0.00
3000	5	2765.13	896	1007.0	1176	31.19
3000	10	2456.81	781	878.6	950	38.86
3000	25	2004.51	615	682.8	764	50.12
3000	50	1702.02	516	547.2	574	57.64
3000	100	1370.96	332	419.8	474	65.88
3000	200	1277.18	261	327.0	360	68.22
3000	500	1525.36	206	228.0	248	62.04
3000	1000	3728.50	172	180.5	195	7.21
3000	2000	14920.00	141	147.8	150	-271.30

Table 7: Static multicut aggregation for SSN

Sample Size	Num aggs	CPU sec	Min Iters	Aver Iters	Max Iters	% CPU Reduction
1000	1	1969.52	2569	2767.0	2961	0.00
1000	5	634.58	785	811.4	850	67.78
1000	10	346.65	403	429.0	456	82.40
1000	25	167.07	180	189.8	197	91.52
1000	50	96.41	99	102.6	109	95.10
1000	100	68.21	62	63.6	65	96.54
1000	200	56.99	43	43.8	45	97.11
1000	500	58.43	29	30.2	31	97.03
1000	1000	76.77	24	25.2	27	96.10
2000	1	4004.00	2690	2856.0	2980	0.00
2000	5	1414.22	912	939.0	968	64.68
2000	10	840.35	504	536.6	551	79.01
2000	25	415.28	240	247.0	258	89.63
2000	50	246.87	135	137.0	139	93.83
2000	100	172.25	84	85.0	87	95.70
2000	200	139.63	57	58.0	59	96.51
2000	500	132.74	37	38.0	40	96.68
2000	1000	149.73	29	29.0	29	96.26
2000	2000	229.89	24	24.6	26	94.26
3000	1	6109.21	2693	2935.0	3236	0.00
3000	5	2349.93	983	1059.6	1118	61.53
3000	10	1418.19	572	616.2	648	76.79
3000	25	719.81	280	292.6	302	88.22
3000	50	425.63	158	162.2	167	93.03
3000	100	289.04	98	100.0	103	95.27
3000	200	229.88	67	67.2	68	96.24
3000	500	206.18	41	42.4	44	96.62
3000	1000	239.42	31	32.8	34	96.08
3000	2000	361.58	26	26.8	27	94.08
3000	3000	542.95	24	24.4	25	91.11

Table 8: Static multicut aggregation for STORM

Sample Size	Num aggs	CPU sec	Min Iters	Aver Iters	Max Iters	% CPU Reduction
1000	1	6.54	20	21.8	24	0.00
1000	5	6.50	20	22.0	24	0.61
1000	10	5.84	18	20.2	22	10.71
1000	25	5.72	17	19.4	21	12.50
1000	50	6.52	20	21.4	23	0.28
1000	100	6.09	19	19.0	19	6.85
1000	200	6.69	18	19.2	22	-2.33
1000	500	8.63	17	17.0	17	-31.87
1000	1000	16.43	18	18.4	19	-151.03
2000	1	13.26	20	22.2	24	0.00
2000	5	13.73	22	23.0	24	-3.54
2000	10	12.87	18	22.4	24	2.92
2000	25	12.31	18	21.2	23	7.16
2000	50	12.91	18	21.0	23	2.61
2000	100	12.16	19	20.0	22	8.25
2000	200	12.42	19	19.0	19	6.31
2000	500	18.04	19	20.6	21	-36.05
2000	1000	21.11	17	17.0	17	-59.24
3000	1	20.24	20	22.0	24	0.00
3000	5	18.28	18	21.8	24	9.68
3000	10	18.69	19	21.8	24	7.62
3000	25	17.97	18	20.8	24	11.18
3000	50	19.76	18	22.0	24	2.38
3000	100	20.75	20	22.2	24	-2.52
3000	200	18.27	19	19.4	21	9.70
3000	500	21.68	17	18.6	19	-7.13
3000	1000	32.45	19	19.8	20	-60.32
3000	2000	56.82	18	18.8	19	-180.74
3000	3000	91.86	18	18.8	19	-353.83
5000	1	35.35	23	23.2	24	0.00
5000	5	33.13	18	22.6	24	6.28
5000	10	33.68	20	23.0	24	4.74
5000	25	32.20	19	22.0	24	8.92
5000	50	29.78	19	20.8	23	15.76
5000	100	32.49	19	21.0	23	8.09
5000	200	33.90	19	21.8	23	4.08
5000	500	33.80	19	19.0	19	4.41
5000	1000	47.47	19	20.6	22	-34.27
5000	2000	78.36	18	19.6	21	-121.63
5000	5000	229.00	18	18.6	19	-547.67
10000	1	70.55	23	23.2	24	0.00
10000	5	70.35	23	23.2	24	0.28
10000	10	70.67	22	23.4	24	-0.17
10000	25	71.60	23	23.8	24	-1.48
10000	50	69.74	23	23.0	23	1.15
10000	100	51.11	18	18.8	19	27.55
10000	200	64.06	19	20.6	22	9.20
10000	500	68.81	19	20.2	22	2.47
10000	1000	74.46	19	19.0	19	-5.55
10000	2000	114.06	20	20.8	21	-61.67

References

Benders, J. F.: 1962, Partitioning procedures for solving mixed-variable programming problems, *Numerische Mathematik* **54**, 238–252.

Birge, J.: 1985, Aggregation bounds in stochastic linear programming, *Mathematical Programming* **31**, 25–41.

Birge, J.: 1988, A multicut algorithm for two-stage stochastic linear programs, *European Journal of Operational Research* **34**, 384–392.

Birge, J. R. and Louveaux, F. V.: 1997, *Introduction to Stochastic Programming*, Springer, New York.

Gassmann, H. I.: 1990, MSLiP: a computer code for the multistage stochastic linear programming problem, *Mathematical Programming* **47**, 407–423.

Higle, J. and Sen, S.: 1991, Stochastic decomposition: An algorithm for two-stage stochastic linear programs with recourse, *Mathematics of Operational Research* **16**, 650–669.

Higle, J. and Sen, S.: 1996, *Stochastic Decomposition: A Statistical Method for Large Scale stochastic linear programming*, Kluwer Academic Publisher, Dordrecht.

Higle, J. and Zhao, L.: 2007, Adaptive and nonadaptive samples in solving stochastic linear programs: A computational investigation, *Submitted to Computational Optimization and Applications* .

ILOG, I.: 2003, *CPLEX 9.0 Reference Manual*, ILOG CPLEX Division.

Linderoth, J. T. and Wright, S. J.: 2003, Decomposition algorithms for stochastic programming on a computational grid, *Computational Optimization and Applications* **24**, 207–250.

Mak, W. K., Morton, D. and Wood., R.: 1999, Monte carlo bounding techniques for determining solution quality in stochastic programs, *Operations Research Letters* **24**, 47–56.

Mulvey, J. M. and Ruszczynski, A.: 1992, A new scenario decomposition method for large scale stochastic optimization, *Technical report sor-91-19*, Dept. of Civil Engineering and Operations Research, Princeton Univ. Princeton, N.J. 08544.

Ruszczynski, A.: 1988, A regularized decomposition for minimizing a sum of polyhedral functions, *Mathematical Programming* **35**, 309–333.

Sen, S., Doverspike, R. and Cosares, S.: 1994, Network planning with random demand, *Telecommunications Systems* **3**, 11–30.

Shapiro, A.: 2000, Stochastic programming by Monte Carlo simulation methods, *Available at http://www2.isye.gatech.edu/∼ashapiro/publications.html*.
URL: *http://www2.isye.gatech.edu/∼ashapiro/publications.html*

Van Slyke, R. and Wets, R.-B.: 1969, L-shaped linear programs with application to optimal control and stochastic programming, *SIAM Journal on Applied Mathematics* **17**, 638–663.