Stochastics and Statistics

Adaptive multicut aggregation for two-stage stochastic linear programs with recourse

Svyatoslav Trukhanov a, Lewis Ntaimo a, Andrew Schaefer b

a Department of Industrial and Systems Engineering, Texas A&M University, 3131 TAMU, College Station, TX 77843, USA
b Department of Industrial Engineering, 1048 Benedum Hall, University of Pittsburgh, Pittsburgh, PA 15261, USA

1. Introduction

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 (Birge and Louveaux, 1988) 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 trade-offs in terms of computational time are 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. Computational results reveal that a cut aggregation level that is between the single cut and the multicut can result in substantial computational savings over the single cut method.

The contributions of this paper include an adaptive optimality multicut method for SLP, an implementation of the algorithm in C++ using the CPLEX Callable Library (ILOG, 2003) and a computational investigation with instances from the literature. These include the SSN (Sen et al., 1994) with $10^5$ scenarios and STORM (Mulvey and Ruszczyński, 1995) with $10^7$ scenarios. The authors reported CPU time improvements of up to 83.5% for instances of SSN with $10^6$ scenarios.

The relative advantages of the single cut and multicut methods have already been explored (Birge and Louveaux, 1988, 1990). Citing earlier work (Birge and Louveaux, 1988, 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. Birge (1985) used variable aggregation to reduce the problem size and to find upper and lower bounds on the optimal solution. Ruszczyński (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 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 Ruszczyński, 1995) with $10^7$ scenarios. The authors reported CPU time improvements of up to 83.5% for instances of SSN with $10^6$ scenarios.

The contributions of this paper include an adaptive optimality multicut method for SLP, an implementation of the algorithm in C++ using the CPLEX Callable Library (ILOG, 2003) and a computational investigation with 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 sample average approximation method (Shapiro, 2003), the trust-region method (Linderoth and Wright, 2003) and the stochastic decomposition method (Higle and Sen, 1991), may benefit.
Table 1
Problem characteristics.

<table>
<thead>
<tr>
<th>Problem</th>
<th>First stage</th>
<th>Second stage</th>
<th>No. of random variables</th>
<th>No. of scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rows</td>
<td>Cols</td>
<td>Rows</td>
<td>Cols</td>
</tr>
<tr>
<td>20TERM</td>
<td>3</td>
<td>63</td>
<td>124</td>
<td>764</td>
</tr>
<tr>
<td>SSN</td>
<td>1</td>
<td>89</td>
<td>175</td>
<td>706</td>
</tr>
<tr>
<td>STORM</td>
<td>59</td>
<td>128</td>
<td>526</td>
<td>1259</td>
</tr>
</tbody>
</table>

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

Fig. 2. Adaptive multicut aggregation % CPU improvement over regular L-shaped method for SSN.
from the adaptive multicut approach. Furthermore, the adaptive optimality multicut method we propose is relatively easier to implement than the trust-region method and provides significant improvement in computational time for the instance classes we considered.

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:

\[ \text{Min } c^\top x + \sum_{s \in S} p_i q_{i,s} y_s \]  
\[ \text{s.t. } Ax = b \]  
\[ T x + W y_s \geq r_s, \forall s \in S \]  
\[ x \geq 0, y_s \geq 0, \forall s \in S \]
where $x \in \mathbb{R}^n$ is the vector of first stage decision variables with the corresponding cost vector $c \in \mathbb{R}^n; A \in \mathbb{R}^{m \times n}$ is the first stage constraint matrix; and $b \in \mathbb{R}^m$ is the first stage right-hand side vector. Additionally, $S$ is the set of scenarios (outcomes), and for each scenario $s \in S, y_s \in \mathbb{R}^n$ is the vector of second stage decision variables; $p_s$ is the probability of occurrence for scenario $s; q_s \in \mathbb{R}^n$ is the second stage cost vector; $r_s \in \mathbb{R}^{m_s \times n}$ is the second stage right-hand side vector; $T_s \in \mathbb{R}^{m_s \times m}$ is the technology matrix; and $W \in \mathbb{R}^{m_s \times n}$ 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 (1d).

In general, the single cut L-shaped method tends 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.
Table 2
Adaptive multicut aggregation for 20TERM.

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

Fig. 7. Static multicut aggregation % CPU improvement over regular L-shaped method for STORM.
Our goal is to devise an algorithm that performs better than the single cut L-shaped and the multifacit 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 multifacit method with ‘partial’ cut aggregation. This means that such an algorithm requires partitioning the sample space S into D aggregates (Sd) with such that S = S1 ∪ S2 ∪ ... ∪ SD and Sd ∩ Sj = ∅ ∀ i ≠ j. Then at the initialization step of the algorithm, optimality cut variables θd are introduced, one for each aggregate. The optimality cuts are also generated one per aggregate in the form (βd^T)x + θd ≥ x_d^T, where (βd^T) = ∑_{i∈Sd} pi(πi^T) T_i and x_d^T = ∑_{i∈Sd} pi(πi^T) r_i.

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 Sd = {Sd}, d = 1, ..., |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 S1 = 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 run-time is achieved on some middle level of aggregation 1 < D < |S| but this level is not known a priori. For two out of the three classes of instances we tested, the proposed adaptive multicut method outperformed both the pure single cut and the multicut method.

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 cut variables, that is, the number of optimality cut 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 cut variables in the master problem decreases. If |S| is not ‘very large’ one could initialize the algorithm as pure multicut, where an optimality cut variable θd is created for each s ∈ 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 kth iteration let the scenario set partition be ℒ(k) = {S1, S2, ..., Sk}, where the sets Si, i = 1, ..., k, are disjoint, that is, Si ∩ Sj = ∅ ∀ i ≠ j and ∪_{i=1}^k Si = S. Therefore, each scenario s ∈ S belongs to only one Si ∈ ℒ(k) and the aggregate probability of Si is defined as Yi = ∑_{i∈Si} pi with ∑_{s∈S} Si = 1. Let ℒ(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:

\[ \text{Min } c^T x + \sum_{d∈ℒ(k)} \theta_d, \]  
\[ \text{s.t. } Ax = b, \]  
\[ \langle β_1^d \rangle^T x ≥ x^t, t ∈ \{1, ..., k\} \setminus ℒ(k), \]  
\[ \langle β_2^d \rangle^T x + \theta_d ≥ x^t, t ∈ ℒ(k), d ∈ ℒ(k), \]  
\[ x ≥ 0, \]

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 = 0, ℒ(0) = ∅, and initialize ℒ(0).
Step 1: Solve the Master Problem.
Set k = k + 1 and solve problem (2). Let \((x^k, \text{vec}(\beta^d))_{d∈ℒ(k)}\) be an optimal solution to problem (2). If no constraint (2d) is present for some \(d ∈ ℒ(k)\), \(θ_d\) is set equal to \(-∞\) and is ignored in the computation.

Step 2: Update Cut Aggregation Level.
Step 2a Cut Aggregation.
Generate aggregate ℒ(k) using ℒ(k − 1) based on some aggregation scheme. Each element of ℒ(k) is a union of some elements from ℒ(k − 1). If, according to the aggregation scheme, \(d_1, d_2, ..., d_p ∈ ℒ(k − 1)\) are aggregated into \(d ∈ ℒ(k)\), then \(d = \bigcup_{i=1}^p d_i\) and \(p_d = \sum_{i=1}^p p_{d_i}\). Master problem (2) will be modified by removing variables \(θ_{d_1}, ..., θ_{d_p}\) and introducing a 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
\[ x_d^T = p_d \sum_{t=1}^T (1/p_d) x_{d_t} \quad ∀ d ∈ ℒ(k), t ∈ ℒ(k), \]  
\[ β_d^T = p_d \sum_{t=1}^T (1/p_d) β_{d_t} \quad ∀ d ∈ ℒ(k), t ∈ ℒ(k). \]

For all iterations \(t ∈ ℒ(k)\) replace cuts corresponding to \(d_1, ..., d_i\) with one new cut \(⟨ β_d^T ⟩^T x + θ_d ≥ x_d^T \) corresponding to \(d\).

Step 3: Solve Scenario Subproblems.
For all \(s ∈ S\) solve:

\[ \text{Min } q^s_1 y, \]  
\[ \text{s.t. } Wy ≥ r_s − T_s x^k, \]  
\[ y ≥ 0. \]

Let \(π^s_k\) be the dual multipliers associated with an optimal solution of problem (5c). For each \(d ∈ ℒ(k)\) if
\[ \theta_d < p_d \sum_{s∈S} π^s_k (r_s − T_s x^k), \]

define
\[ x_d^T = \sum_{s∈S} p_s (π^s_k)^T r_s, \]  

and
\[ ⟨ β_d^T ⟩^T = \sum_{s∈S} p_s (π^s_k)^T T_s. \]

If condition (6) does not hold for any \(d ∈ ℒ(k)\), stop \(x^k\) is optimal. Otherwise, if for all \(s ∈ S\) subproblem (5c) is optimal, put \(ℒ(k) = ℒ(k − 1) \cup \{k\}\) and go to Step 1. If problem (5c) is infeasible for some \(s ∈ S\), let \(μ_s\) be the associated dual extreme ray and define
\[ x^k = μ_s^T r_s, \]  

and
\[ ⟨ β_d^T ⟩^T = μ_s^T T_s. \]

Go to Step 1.
Notice that in Basic Adaptive Multicut algorithm optimality cuts are only generated and added to the master program when all scenario subproblems (5c) are feasible. Otherwise, when an infeasible subproblem is encountered, a feasibility cut based on that subproblem is generated and added to the master program. An alternative implementation is to find all infeasible subproblems and generate a feasibility cut from each one to add to the master program. Also, optimality cuts can be generated for all \( d \in \mathcal{S}(k) \) such that all subproblems \( s \in d \) are feasible. Since convergence of Basic Adaptive Multicut 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 scheme that one can use to obtain \( \mathcal{S}(k) \) from \( \mathcal{S}(k-1) \).

There are several possibilities one can think of when it comes to aggregation schemes, but our experience showed that most do not perform well. We tried several schemes and through computational experimentation devised a scheme that worked the best. This aggregation scheme involves two basic rules, redundancy threshold and bound on the number of aggregates, and can be described as follows:

**Aggregation Scheme:**

- **Redundancy Threshold** \((\delta)\). 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 \( d \in \mathcal{S}(k) \). Let \( f_d \) be the number of iterations when all optimality cuts corresponding to \( d \) are redundant, that is, the corresponding dual multipliers are zero. Also let \( 0 < \delta < 1 \) be a given threshold. Then all \( d \) such that \( f_d / |\mathcal{S}(k)| > \delta \) are combined to form one aggregate. This means that the optimality cuts obtained from aggregate \( d \) are often redundant at least \( \delta \) 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 is necessary to prevent aggregating all scenarios prematurely, leading to the single cut method. Similarly, a bound on the maximum number of aggregates can also be imposed to curtail the proliferation of optimality cuts in the master program. These bounds have to be set \textit{a priori} and kept constant throughout the algorithm run.

We should point out that even though in Redundancy Threshold we require for each \( d \in \mathcal{S}(k) \) all the optimality cuts to be redundant, one can consider an implementation where only a fraction of the optimality cuts are required to be redundant.

Another aggregation scheme we tried is to aggregate scenarios that yield "similar" optimality cuts, that is, cuts with similar gradients (orientation). We implemented this scheme but did not observe significant gains in computation time due to the additional computational burden required for determining optimality cuts that are similar. This task involves comparing thousands of cut coefficients for thousands of all cuts generated in the course of the algorithm run. We also 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. Remember that deletion of hundreds of cuts at a time from the master program can also be a time consuming process.

An opposite approach to cut aggregation is \textit{cut disaggregation}, that is, partitioning an aggregate into two or more aggregates. This allows for more cut information about the recourse function to be available in the master program. However, the major drawback to this scheme is that all cut information has to be kept in memory (or at least written to file) in order to partition any aggregate. We tried this scheme and did not obtain any promising results based on the instances we used. This scheme requires careful book-keeping of all the generated cuts. Consequently, in our implementation this resulted in a computer memory intensive scheme. We encountered several memory and algorithm speed-up issues related to (1) the storage of cut information (e.g. cut coefficients, right-hand sides, iteration index, and scenario index); (2) retrieval and aggregation of cut information; (3) deletion of cut information from the master program; and (4) increasing the master program size. These issues resulted in a significantly slow algorithm, defeating the benefits of the adaptive approach. Nevertheless, we believe that cut disaggregation still needs further research, especially with an advanced implementation of the cut disaggregation scheme to alleviate the above-listed computational issues.

Finally, we should point out that even though we did not experiment with sampling-based aggregation approaches, one can still incorporate such approaches into the proposed framework.
example, one can aggregate scenarios based on importance sampling (Infanger, 1992). Another approach would be to use an accelerated cutting plane method such as regularized decomposition (Ruszczynski, 1988) in conjunction with the proposed cut aggregation scheme.

4. Computational results

We implemented the adaptive multicut algorithm in C++ using the CPLEX Callable Library (ILOG, 2003) 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, 1995). The problem characteristics of these instances are given in Table 1.

Due to the extremely large number of scenarios, solving these instances with the single cut 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 10,000 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 were recorded. All the experiments were conducted on an Intel Pentium 4 3.2 GHz processor with 512 MB 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 a 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 of 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.

<table>
<thead>
<tr>
<th>Sample size</th>
<th>Agg-max</th>
<th>CPU seconds</th>
<th>Min iters</th>
<th>Aver iters</th>
<th>Max iters</th>
<th>% CPU reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10</td>
<td>21.62</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–230.28</td>
</tr>
<tr>
<td>1000</td>
<td>20</td>
<td>21.07</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–221.82</td>
</tr>
<tr>
<td>1000</td>
<td>30</td>
<td>21.27</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–224.88</td>
</tr>
<tr>
<td>1000</td>
<td>40</td>
<td>21.20</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–223.85</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>21.28</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–225.09</td>
</tr>
<tr>
<td>1000</td>
<td>60</td>
<td>20.82</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–217.95</td>
</tr>
<tr>
<td>1000</td>
<td>70</td>
<td>20.87</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–218.72</td>
</tr>
<tr>
<td>1000</td>
<td>80</td>
<td>21.08</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–221.97</td>
</tr>
<tr>
<td>1000</td>
<td>90</td>
<td>20.65</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–215.40</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>20.87</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–218.75</td>
</tr>
<tr>
<td>1000</td>
<td>200</td>
<td>20.79</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–217.62</td>
</tr>
<tr>
<td>1000</td>
<td>300</td>
<td>20.61</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–214.81</td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>20.65</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–215.39</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>20.61</td>
<td>18</td>
<td>18.4</td>
<td>19</td>
<td>–214.75</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>21.56</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–62.53</td>
</tr>
<tr>
<td>2000</td>
<td>20</td>
<td>21.47</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.85</td>
</tr>
<tr>
<td>2000</td>
<td>30</td>
<td>21.36</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.08</td>
</tr>
<tr>
<td>2000</td>
<td>40</td>
<td>21.39</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.26</td>
</tr>
<tr>
<td>2000</td>
<td>50</td>
<td>21.43</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.57</td>
</tr>
<tr>
<td>2000</td>
<td>60</td>
<td>21.61</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–62.91</td>
</tr>
<tr>
<td>2000</td>
<td>70</td>
<td>21.43</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.57</td>
</tr>
<tr>
<td>2000</td>
<td>80</td>
<td>21.39</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.29</td>
</tr>
<tr>
<td>2000</td>
<td>90</td>
<td>21.42</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.50</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>21.41</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.45</td>
</tr>
<tr>
<td>2000</td>
<td>200</td>
<td>21.41</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.40</td>
</tr>
<tr>
<td>2000</td>
<td>300</td>
<td>21.37</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.14</td>
</tr>
<tr>
<td>2000</td>
<td>500</td>
<td>21.39</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.31</td>
</tr>
<tr>
<td>2000</td>
<td>1000</td>
<td>21.41</td>
<td>17</td>
<td>17.0</td>
<td>17</td>
<td>–61.45</td>
</tr>
<tr>
<td>3000</td>
<td>10</td>
<td>48.23</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–138.26</td>
</tr>
<tr>
<td>3000</td>
<td>20</td>
<td>48.57</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–139.98</td>
</tr>
<tr>
<td>3000</td>
<td>30</td>
<td>48.86</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–141.37</td>
</tr>
<tr>
<td>3000</td>
<td>40</td>
<td>48.22</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–138.25</td>
</tr>
<tr>
<td>3000</td>
<td>50</td>
<td>47.46</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–134.47</td>
</tr>
<tr>
<td>3000</td>
<td>60</td>
<td>48.24</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–138.31</td>
</tr>
<tr>
<td>3000</td>
<td>70</td>
<td>48.12</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–137.72</td>
</tr>
<tr>
<td>3000</td>
<td>80</td>
<td>47.62</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–135.28</td>
</tr>
<tr>
<td>3000</td>
<td>90</td>
<td>48.40</td>
<td>20</td>
<td>20.0</td>
<td>20</td>
<td>–139.11</td>
</tr>
<tr>
<td>3000</td>
<td>100</td>
<td>46.97</td>
<td>19</td>
<td>19.8</td>
<td>20</td>
<td>–132.06</td>
</tr>
<tr>
<td>3000</td>
<td>200</td>
<td>46.90</td>
<td>20</td>
<td>20.0</td>
<td>20</td>
<td>–131.69</td>
</tr>
<tr>
<td>3000</td>
<td>300</td>
<td>46.09</td>
<td>20</td>
<td>20.0</td>
<td>20</td>
<td>–127.71</td>
</tr>
<tr>
<td>3000</td>
<td>500</td>
<td>46.06</td>
<td>20</td>
<td>20.0</td>
<td>20</td>
<td>–127.57</td>
</tr>
<tr>
<td>3000</td>
<td>1000</td>
<td>46.11</td>
<td>20</td>
<td>20.0</td>
<td>20</td>
<td>–127.82</td>
</tr>
</tbody>
</table>
Table 5
Adaptive multicut aggregation with redundancy threshold $\delta$.

<table>
<thead>
<tr>
<th>Problem name</th>
<th>Agg-max</th>
<th>CPU Time (seconds)</th>
<th>Aver iters</th>
<th>% CPU reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5</td>
<td>0.6</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>20term</td>
<td>10</td>
<td>407.01</td>
<td>402.15</td>
<td>384.90</td>
</tr>
<tr>
<td>20term</td>
<td>20</td>
<td>450.09</td>
<td>434.86</td>
<td>470.50</td>
</tr>
<tr>
<td>20term</td>
<td>30</td>
<td>522.31</td>
<td>543.72</td>
<td>490.48</td>
</tr>
<tr>
<td>20term</td>
<td>40</td>
<td>593.87</td>
<td>596.54</td>
<td>564.78</td>
</tr>
<tr>
<td>20term</td>
<td>50</td>
<td>512.02</td>
<td>518.50</td>
<td>612.98</td>
</tr>
<tr>
<td>20term</td>
<td>60</td>
<td>652.42</td>
<td>629.03</td>
<td>620.88</td>
</tr>
<tr>
<td>20term</td>
<td>70</td>
<td>630.61</td>
<td>634.00</td>
<td>603.24</td>
</tr>
<tr>
<td>20term</td>
<td>80</td>
<td>618.35</td>
<td>630.86</td>
<td>619.25</td>
</tr>
<tr>
<td>20term</td>
<td>90</td>
<td>551.51</td>
<td>588.42</td>
<td>666.14</td>
</tr>
<tr>
<td>20term</td>
<td>100</td>
<td>723.23</td>
<td>708.96</td>
<td>738.51</td>
</tr>
</tbody>
</table>

Sample size | Num aggs | CPU seconds | 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      | 60       | 237.44      | 235.90    | 235.76     | 262.2     | 87.31          |
| 1000      | 70       | 267.75      | 262.42    | 262.49     | 296.4     | 85.89          |
| 1000      | 80       | 301.38      | 298.50    | 298.74     | 339.0     | 83.90          |
| 1000      | 90       | 315.24      | 323.30    | 318.39     | 360.2     | 83.16          |
| 1000      | 100      | 354.43      | 361.52    | 358.93     | 412.4     | 81.06          |

Storm      | 10       | 11.76       | 10.87     | 11.15      | 21.4      | 18.4           |
| Storm      | 20       | 10.90       | 10.54     | 10.67      | 19.2      | 18.4           |
| Storm      | 30       | 10.81       | 10.34     | 10.57      | 19.0      | 18.4           |
| Storm      | 40       | 11.25       | 10.32     | 10.47      | 21.0      | 18.4           |
| Storm      | 50       | 11.20       | 10.30     | 10.46      | 20.8      | 18.4           |
| Storm      | 60       | 11.72       | 10.32     | 10.48      | 22.8      | 18.4           |
| Storm      | 70       | 11.81       | 10.29     | 10.41      | 23.0      | 18.4           |
| Storm      | 80       | 11.76       | 10.30     | 10.47      | 22.8      | 18.4           |
| Storm      | 90       | 11.64       | 10.20     | 10.49      | 22.2      | 18.0           |
| Storm      | 100      | 11.94       | 10.29     | 10.42      | 23.2      | 18.4           |

Table 6
Static multicut aggregation for 20TERM.

<table>
<thead>
<tr>
<th>Sample size</th>
<th>Num aggs</th>
<th>CPU seconds</th>
<th>Min iters</th>
<th>Aver iters</th>
<th>Max iters</th>
<th>% CPU reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>1</td>
<td>2168.89</td>
<td>1082</td>
<td>1232.6</td>
<td>1480</td>
<td>0.00</td>
</tr>
<tr>
<td>2000</td>
<td>5</td>
<td>1804.45</td>
<td>830</td>
<td>976.2</td>
<td>1054</td>
<td>16.80</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>1867.90</td>
<td>666</td>
<td>853.6</td>
<td>919</td>
<td>22.18</td>
</tr>
<tr>
<td>2000</td>
<td>25</td>
<td>1294.14</td>
<td>611</td>
<td>643.0</td>
<td>684</td>
<td>40.33</td>
</tr>
<tr>
<td>2000</td>
<td>50</td>
<td>1018.93</td>
<td>448</td>
<td>483.8</td>
<td>523</td>
<td>53.02</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>844.92</td>
<td>350</td>
<td>372.8</td>
<td>388</td>
<td>61.04</td>
</tr>
<tr>
<td>2000</td>
<td>200</td>
<td>781.94</td>
<td>241</td>
<td>283.6</td>
<td>302</td>
<td>63.95</td>
</tr>
<tr>
<td>2000</td>
<td>500</td>
<td>1037.75</td>
<td>182</td>
<td>194.4</td>
<td>201</td>
<td>52.15</td>
</tr>
<tr>
<td>2000</td>
<td>1000</td>
<td>2351.56</td>
<td>139</td>
<td>154.0</td>
<td>166</td>
<td>7.82</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
<td>12065.45</td>
<td>114</td>
<td>1268.0</td>
<td>136</td>
<td>456.30</td>
</tr>
</tbody>
</table>
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 \textit{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 $1, m + 1, 2m + 1, \ldots$, were aggregated to aggregate 1, cuts $2, m + 2, 2m + 2, \ldots$, were aggregated to aggregate 2, and so on.

The computational results are summarized in the plots in Figs. 5 and 6. In the plots ‘% reduction in CPU time’ is relative to the single cut L-shaped method, which acted as the benchmark. The results in Fig. 5 show that a level of aggregation between single cut and pure multicut for 20TERM, results in about 60% reduction in CPU time over the L-shaped for all the instances. For SSN (Fig. 6) percent improvements over 90% are achieved. The results for STORM (Fig. 7) showed improvements of just over 10%, which provides a bound on the performance of the adaptive multicut method, which showed no aggregation being the best level of aggregation for this instance. 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.

In Fig. 6 (SSN) the percentage reduction in CPU time increases with the number of aggregates and then tapers off after 200 aggregates. On the contrary, in Fig. 7 (STORM) we see that the percentage reduction in CPU time first remains steady up to about 200 aggregates, and then decreases steadily. This contrasting behavior can be attributed to the different shapes of the expected recourse function in each instance. For example, based on our computational experimentation, STORM seems to have a fairly ‘flat’ expected recourse function around the optimal solution and thus does not benefit much from a large number of aggregates. On the contrary, the expected recourse function in SSN is not as ‘flat’ around the optimal solution and thus benefits from a large number of aggregates (more information).

The computational results with static multicut aggregation apply 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 \textit{a priori}. Therefore, the adaptive multicut method provides a viable approach. Furthermore, this method is relatively easier to implement and provides similar performance as the trust-region method of Linderoth and Wright (2003).

### Table 7

<table>
<thead>
<tr>
<th>Sample size</th>
<th>Num aggs</th>
<th>CPU seconds</th>
<th>Min iters</th>
<th>Aver iters</th>
<th>Max iters</th>
<th>% CPU reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>1969.52</td>
<td>2569</td>
<td>2767.0</td>
<td>2961</td>
<td>0.00</td>
</tr>
<tr>
<td>1000</td>
<td>5</td>
<td>634.58</td>
<td>785</td>
<td>811.4</td>
<td>850</td>
<td>67.78</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>346.65</td>
<td>403</td>
<td>429.0</td>
<td>456</td>
<td>82.40</td>
</tr>
<tr>
<td>1000</td>
<td>25</td>
<td>167.07</td>
<td>180</td>
<td>189.8</td>
<td>197</td>
<td>91.52</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>96.41</td>
<td>99</td>
<td>102.6</td>
<td>109</td>
<td>95.10</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>68.21</td>
<td>62</td>
<td>63.6</td>
<td>65</td>
<td>95.54</td>
</tr>
<tr>
<td>1000</td>
<td>200</td>
<td>56.99</td>
<td>43</td>
<td>43.8</td>
<td>45</td>
<td>97.11</td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>58.43</td>
<td>29</td>
<td>30.2</td>
<td>31</td>
<td>97.03</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>76.77</td>
<td>24</td>
<td>25.2</td>
<td>27</td>
<td>96.10</td>
</tr>
<tr>
<td>2000</td>
<td>1</td>
<td>4004.00</td>
<td>2690</td>
<td>2856.0</td>
<td>2980</td>
<td>0.00</td>
</tr>
<tr>
<td>2000</td>
<td>5</td>
<td>1414.22</td>
<td>912</td>
<td>939.0</td>
<td>968</td>
<td>64.68</td>
</tr>
<tr>
<td>2000</td>
<td>10</td>
<td>840.35</td>
<td>504</td>
<td>536.6</td>
<td>551</td>
<td>79.01</td>
</tr>
<tr>
<td>2000</td>
<td>25</td>
<td>415.28</td>
<td>240</td>
<td>247.0</td>
<td>258</td>
<td>89.63</td>
</tr>
<tr>
<td>2000</td>
<td>50</td>
<td>246.87</td>
<td>135</td>
<td>137.0</td>
<td>139</td>
<td>93.83</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>172.25</td>
<td>84</td>
<td>85.0</td>
<td>87</td>
<td>95.70</td>
</tr>
<tr>
<td>2000</td>
<td>200</td>
<td>139.63</td>
<td>57</td>
<td>58.0</td>
<td>59</td>
<td>96.51</td>
</tr>
<tr>
<td>2000</td>
<td>500</td>
<td>132.74</td>
<td>37</td>
<td>38.0</td>
<td>40</td>
<td>96.68</td>
</tr>
<tr>
<td>2000</td>
<td>1000</td>
<td>149.73</td>
<td>29</td>
<td>29.0</td>
<td>29</td>
<td>96.26</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
<td>229.89</td>
<td>24</td>
<td>24.6</td>
<td>26</td>
<td>94.26</td>
</tr>
<tr>
<td>3000</td>
<td>1</td>
<td>6109.21</td>
<td>2693</td>
<td>2935.0</td>
<td>3236</td>
<td>0.00</td>
</tr>
<tr>
<td>3000</td>
<td>5</td>
<td>2349.93</td>
<td>983</td>
<td>1059.6</td>
<td>1118</td>
<td>61.53</td>
</tr>
<tr>
<td>3000</td>
<td>10</td>
<td>1418.19</td>
<td>572</td>
<td>616.2</td>
<td>648</td>
<td>76.79</td>
</tr>
<tr>
<td>3000</td>
<td>25</td>
<td>719.81</td>
<td>280</td>
<td>292.6</td>
<td>302</td>
<td>88.22</td>
</tr>
<tr>
<td>3000</td>
<td>50</td>
<td>425.63</td>
<td>158</td>
<td>162.2</td>
<td>167</td>
<td>93.03</td>
</tr>
<tr>
<td>3000</td>
<td>100</td>
<td>289.04</td>
<td>98</td>
<td>100.0</td>
<td>103</td>
<td>95.27</td>
</tr>
<tr>
<td>3000</td>
<td>200</td>
<td>229.88</td>
<td>67</td>
<td>67.2</td>
<td>68</td>
<td>96.24</td>
</tr>
<tr>
<td>3000</td>
<td>500</td>
<td>206.18</td>
<td>41</td>
<td>42.4</td>
<td>44</td>
<td>96.62</td>
</tr>
<tr>
<td>3000</td>
<td>1000</td>
<td>235.42</td>
<td>31</td>
<td>32.8</td>
<td>34</td>
<td>96.08</td>
</tr>
<tr>
<td>3000</td>
<td>2000</td>
<td>361.58</td>
<td>26</td>
<td>26.8</td>
<td>27</td>
<td>94.08</td>
</tr>
<tr>
<td>3000</td>
<td>3000</td>
<td>542.95</td>
<td>24</td>
<td>24.4</td>
<td>25</td>
<td>91.11</td>
</tr>
</tbody>
</table>
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 schemes 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, 2003), the trust-region method (Linderoth and Wright, 2003), and the stochastic decomposition method (Higle and Sen, 1991, 1996, 2009). Future work along this line of work include incorporating both cut aggregation and disaggregation in the adaptive multicut approach, and extending it to integer first stage and multistage stochastic linear programs.

Acknowledgments

The authors are grateful to the anonymous referees for their useful comments concerning a previous version of the paper. This research was supported in part by grants CNS-0540000, CMII-0355433 and CMII-0620780 from the National Science Foundation.

Appendix A. Numerical results

See Tables 2–8.
References


