Improving the L2 Algorithm by Adding Non-Supporting Bender's Type Cuts

Author: Matthew Tanner
Date: 12/13/05

“On my honor as an aggie, I have neither given nor received unauthorized aid on this academic project”

Signature ____________________________________________________________
I. Abstract
The L2 algorithm is used to solve binary first stage, mixed-integer stochastic programs by computing valid supporting hyperplanes at initial feasible points generated by a master program, and using these hyperplanes to approximate the value function of the mixed integer stochastic second stage. The primary weakness of the algorithm is that it gives weak cuts that only slowly narrow down the feasible region of the master program. In fact, in computational experiments, the L2 often gives the total enumeration of the first stage. This project explores benefits of avoiding total enumeration by adding the Bender’s type L-shaped algorithm cuts to the integer master program.

II. Introduction and Literature Review
As with linear stochastic programs, there are a large variety of useful applications for stochastic mixed integer programs. Stochastic information is useful in situations where decisions need to be made before all information is known, and when the future information is highly random. Stochastic mixed integer programs are necessary when decisions are discrete and suboptimal decisions using rounding are highly expensive. Examples of applications for stochastic mixed integer programming include, facility location for power generation companies, hurricane evacuation plans, and fire suppression plans.

The basic form of a two stage stochastic integer program is given in the following set of equations. The first stage decisions are given by the x variables and need to be made before the realization of the random elements. The second stage decisions are given by the y variables and are made after the realization of the random data. The goal
is to minimize the cost of the first stage decisions plus the expected cost of the second stage decisions.

\[
\min \quad c^T x + E_\omega [f(x, \omega)] \\
\text{subject to} \\
Ax = b \\
x \geq 0 \quad x \text{ integer}
\]

Equation 2.1:

\[f(x, \omega) = \min q^T y\]

Equation 2.2: \quad \text{subject to} \quad Wy = r(\omega) - Tx \\
y \geq 0 \quad y \text{ integer}

The problem with developing algorithms for stochastic mixed integer programs is that they lack many of the “nice” characteristics that allow for good algorithms in other branches of optimization. The basic approach of most stochastic programming algorithms is to approximate the value function of the stochastic subproblems with a series of supporting hyperplanes that guarantee convergence to the optimal first stage/master program solution. For linear stochastic programs, the L-shaped algorithm approximates the second stage value function with the convex combination of Benders algorithm cuts for each subproblem scenario. Unfortunately, this technique relies upon the strong law of duality, which does not hold for integer second stage decisions.

In a 1993 paper, Laporte and Louveaux (1993) propose one of the first successful algorithms for stochastic mixed integer programs. They first describe an extension of the L-shaped method called the integer L-shaped method consisting of the usual L-shaped procedure combined with a branching algorithm whenever the algorithm violates the integrality restrictions. This algorithm is guaranteed to converge in a finite number of
steps, but since the L-shaped algorithm is applied to a possibly exponential number of
nodes, it is not really computationally tractable.

In the same paper, Laporte and Louveaux continue with an algorithm specific to
the case where the first stage decision variables are binary. From here on, this will be
referred to as the L2 algorithm. They derive a supporting hyperplane that can be used at
each feasible binary first stage decision. Their assumptions are that a valid lower bound
L exists for the subproblem value function, and that the value function q_x can be
computed at every feasible point. Their cut is given in equation 2.1.

\[
\theta \geq (Q_x - L) \left( \sum_{i \in S} x_i - \sum_{i \notin S} x_i \right) - (q_x - L)(|S| - 1) + L
\]

This cut is valid and supporting but has a definite weakness in that it is a weak cut. There
is little chance that the cut will give a non-trivial lower bound to any feasible x_k except
for the original point that the cut was made for. The effect of this is that in computational
experiments, this algorithm often involves total enumeration of the first stage variables.

Laporte and Louveaux finish their paper with some suggestions for improved
optimality cuts. They suggest adding relaxed L-shaped cuts, the effects of which are the
scope of this project, and they suggest some improved optimality cuts using the
characteristics of the problem being solved. This new optimality cut involves looking at
the one-neighbors of the master stage solution and deriving an optimal cut using this
extra information gleaned from the bounds of the value function at these neighbors.

There have been several other algorithms proposed to solve these types of
problems. They are put into categories according to what types of problems they can
solve and whether or not they are exact. The most general algorithms are exact algorithms for mixed integer stochastic programs. The faster algorithms seem to be for more specific cases such as binary first stage, or other problem structures that lend themselves to solution. The next part gives a description of some of the algorithms that have been developed recently.

In a 1998 paper, Schultz, Stougie, and Van der Klerk show that since the value function of an integer program is constant over well chosen hyper-rectangles. If there integer variables are bounded then there is only a finite number of such boxes. They proved that mixed integer stochastic programs can be solved by enumerating all of these boxes and then solving the subproblems from this information. The advantage to this method is that it relaxes the need for the pure binary first stages that is needed for the L2 algorithm. The problem is that the number of these boxes is extremely large as the problem size increases and so the algorithm may not be computationally feasible for many problems.

In 2004, Ahmed, Tawarmalani, and Sahinidis describe an improvement upon Shultz’s enumeration algorithm. They are able to preprocess the master problem feasible set so that there is no need to search the entire space. They use the same boxes as Schultz et al. but instead of enumerating all of them, they find them as needed, and fathom them when they are no longer needed. The algorithm is basically a branch and cut algorithm that uses these boxes as nodes in the algorithm. The paper gives a detailed procedure for computing the bounds on the boxes and rules for fathoming and finding new nodes. Computationally, the algorithm is much faster than Shultz et al. as they do not enumerate every one of the boxes.
In a later paper, Ahmed and Shapiro describe a sampling algorithm as an improvement on the above algorithm for stochastic integer programs. The authors use a sample average approximation to compute the value function objective instead of computing the exact value function objective each iteration. This helps avoid the main weakness of the L2 algorithm, which is the need to solve every integer subproblem each iteration of the algorithm. The algorithm is a branch and cut type algorithm like the integer L-shaped that Laporte and Louveaux proposed, so it keeps the same problems of solving an L-shaped algorithm at every node. In their computational results, they found that the algorithm was able to solve large problems to within an acceptable variance quickly in comparison to Cplex 7.0.

One last algorithm that effectively solves pure binary first stage programs is the D2 algorithm developed by Higle and Sen. This algorithm is used for problems that have general mixed-integer subproblems. The algorithm involves adding cuts to both the master and the subproblems in an attempt more quickly reach the optimal master stage and avoid enumerating too many of the value function points. The subproblem is approximated using sequential convexification because this technique can be used to avoid explicitly solving every subproblem scenario during each iteration. The cuts to the master program use disjunctive programming to force the master problem solutions to be integer. The other main result of this paper is the use of common cut coefficients to be able to reuse cuts by only changing the RHS scalar. The assumption needed for this is fixed recourse, but this is not too confining. Computationally, this algorithm has had good success in quickly solving relatively large problems, and the need for the first stage problems to be binary is not too big of an assumption.
In a different paper, Sen and Sherali (2005) propose an extension of the D2 Algorithm called the D2-BAC. The D2-BAC works by relaxing the requirement that subproblems need to be solved to optimality each iteration. Instead, a maximum number of nodes to explore is specified for each attempt to solve a MIP. This prevents the code from getting bogged down in the early iterations of the algorithm when the master solutions are far from optimal. The D2-BAC uses a branch and cut algorithm to ensure integrality in the solution.

Given these different algorithms for solving stochastic mixed integer programs, a big question is which one of the approaches is the most successful. The algorithms of Schultz et al. and Ahmed et al. are both for general integer second stage variables and general first stage variables. The L2 algorithm, which is the focus of this report, is specific to problems with binary first stage variables. For this reason, I will focus on prior work that has been done comparing algorithms for binary first stage variables. The two examples that I have found of these are the L2 algorithm, the D2-BAC algorithm, and the D2 algorithm of Higle and Sen.

In a 2005 paper, Ntaimo and Sen compare the performance of these binary first stage algorithms. Using the sslp test instances, they show that the D2 and the D2-BAC are greatly superior to the L2 in these cases. Mainly because the L2 solves every subproblem to optimality each time, which can take a long time when there are a lot of them. They also show that the L2 totally enumerates the master feasible set each time, unlike the other two algorithms. This project is an attempt to change the L2 to avoid totally enumerating the master feasible set. In the results, I will compare the running
times for solving the test instances with the L-Shaped cuts added in, to the original L2, and also to the results of running the programs with the D2 and the D2-BAC.

III. Solution Approach

The algorithm that I use is the L2 algorithm [1] for binary first stage stochastic mixed integer programs. To improve upon this algorithm, I added in the L-Shaped algorithm optimality cuts[11] after each iteration through the subproblems. The L2 cuts are supporting and will eventually yield the optimal solution, while the L-Shaped cuts are non-supporting and so will not guarantee and optimal solution by themselves. Also, in order for this algorithm to work, it is necessary that the problem has relatively complete recourse. It is also necessary that the subproblem value function has a finite lower bound. I also only implemented the algorithm for problems with all the randomness in the right hand side, but the algorithm can easily be applied to more general stochastic characteristics.

To implement this algorithm, it is necessary to split the problem into sub and master problems. The general form of the split is given in the next set of equations. As was mentioned earlier, the master program corresponds to the first stage decisions and an approximation of the second stage value function, while the subproblems correspond to the second stage decisions made after the realization of the random data. The master program is the same as in equation 2.1, but $E_\omega[f(x, \omega)]$ is replaced with $\theta$, which is used to approximate the expectation according to the optimality cuts being made. The subproblems are exactly the same.

L2 Algorithm with L-Shaped Cuts Added
Step 0:

a) Initialize the upper bound to infinite, the lower bound to negative infinity

b) Set iterations = 0

c) Set value function lower bound L

Note: For this implementation, I set the lower bound by taking the objective value of the recourse function and set \( y(i) = 0 \) if \( q(i) > 0 \), and \( y(i) = 1 \) if \( q(i) < 0 \). This will give a valid lower bound in the case where the subproblem objective is fixed and the variables are binary. If the subproblem object is random, or the subproblem variables are general integer, then it is necessary to compute a relaxation of each individual subproblem value and take the convex combination of those.

d) Solve the initial master program and store the initial master program solution \( x \).

Step 1:

a) For each subproblem, update the RHS according to the stochastic data file

b) Change the RHS using the last \( x \) solution by \( \text{RHS} = r(\omega) - Tx \)

c) Solve each subproblem as a mixed integer program and record the objective value \( Q(x, \omega) \).

d) Relax the subproblem into a linear program and solve to optimality to get the dual solution to the subproblem.

Note: Adding the L-Shaped cuts from the relaxed subproblem is where this algorithm deviates from the basic L2 algorithm. As an implementation issue, it is possible and would be faster to get the dual solutions from Cplex’s solution of the first node in the branch and cut tree, but in this implementation I did not.

Step 2:
a) Compute the Benders cut parameters associated with each subproblem dual solution $\pi_i$.

Equation 3.1: $\alpha_i = \pi_i T, \quad \beta_i = \pi_i r(\omega)$

Step 3:

a) Compute the expectation of all the MIP subproblem values $E[Q(x, \omega)] = Q_x$.
b) Compute the expectation of all the Benders cuts $E[\alpha_i] = \alpha, \quad E[\beta_i] = \beta$
c) Add the constraint given in equation 2.3 using Q to the master program
d) Add the L-Shaped cut using the coefficients computed from the relaxed subproblems to the master program.
e) Compute $U = c^T x + Q_x$ and if U is less than the upper bound, set U as the new upper bound.

Step 4:

a) Solve the updated master program to get the new solution $x$.
b) Set the lower bound equal to the object value of the master program.

Step 5:

a) Check if the upper bound – lower bound is small enough to terminate. If yes, then stop, if no, the return to step 1.

The reason that adding L-shaped cuts improves the speed of the algorithm is because they cut off a larger region of the master space than the L2 algorithms cut, even though they are non-supporting. The L-shaped cuts are still valid because they are an approximation of the value function and feasibility region of the relaxed subproblems,
which obviously includes the feasibility region of the integer subproblems. Using just
those cuts cannot ensure the convergence of the algorithm because of the duality gap.

There are several issues that need to be considered when implementing this
algorithm. The first is that the quality of the cuts improves with the tightness of the lower
bound for the subproblem value function. It would be useful to know how much benefit
a tighter lower bound gives in comparison to the complications of programming it, or the
computation time required. This question is beyond the scope of this project, but I did
close the algorithm computing the lower bound in the way above, with the
performance of the algorithm with the lower bound set to a much lower level and it did
not seem to make much difference except in the confidence interval of the solution at
each iteration.

Another big issue, which is within the scope of this project is the effect of adding
the L-shaped cuts every iteration. The problem is that adding cuts to the master program
increases the size of a mixed-integer problem. After several hundred or a thousand
iterations, the extra time required to solve a mixed integer problem of this size is non-
trivial. Adding two cuts to the master program at a time increases its size even faster.
Also, since the L-shaped cuts are non-supporting, after a certain number of iterations,
they are no longer cutting off any regions of the master program. One experiment that I
will run is to try and measure the effect on running time if the code stops adding the L-
shaped cuts after a certain period of time.

IV. Computational Experiments

I implemented the L2 algorithm with and without the L-shaped cuts added in
using C++ on Microsoft Visual Studio 6.0. Within this program, I used the CPLEX 9.0
callable library to solve the optimization problems that the code generated. I used a computer with a Pentium 4 process with a 2.2 gigahertz chip with 512 megabytes of ram to run all of my experiment. With both experiments, I compare the two algorithms using the number of iterations, the cpu time used, and the also the gap between the optimal solution and the best solution that the program found. I also compare these results with previous algorithms tested on the same instances, with the caveat that the computers used are different and so the results are not directly comparable. I performed my experiments on the standard sslp test set which is a set of problems of varying sizes dealing with the application of stochastic server location. These problems and their optimal solutions can be found at

http://www2.isye.gatech.edu/~sahmed/siplib/sslp/sslp.html.

The first experiment I ran was to test the speed and number of iterations that the L2 algorithm with the L-shaped cuts added takes to solve the problems in the test set, compared with the time and number of iterations that the L2 algorithm alone takes. Table 1 below gives the results of these experiments. The test cases are given in the form sslp_"number of stage 1 variables"_"number of stage 2 variables"_"number of scenarios". The data format is mps format for the core data file, and scenario decomposed smps format for the stochastic data. I allowed the algorithm to run for up to 3200 seconds before I turned it off.
The data from these tests shows that the L2 algorithm generally totally enumerates the first stage decision variables and computes all of the subproblems during each of those iterations. When the L2 cuts are added, the algorithm is generally much more successful at cutting down the number of iterations, and more importantly, even with the increased size of the master program and the extra time needed to solve each subproblem twice, the algorithm is able to solve the test problems much quicker. The most extreme example of this is for sslp_15_45_5 where the L2 algorithm with L-shaped cuts took 6 minutes, while the normal L2 algorithm was unable to solve in the time allotted. This is not too surprising since the L-shaped cuts will help the most in cases where there are too many first stage variables to totally enumerate, and the cost of solving the subproblems twice is not too high.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>Iterations</th>
<th>Master Time (sec)</th>
<th>Sub Time (sec)</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sslp_5_25_50</td>
<td>w/ L-shaped</td>
<td>17</td>
<td>0.14</td>
<td>1.173</td>
<td>1.313</td>
</tr>
<tr>
<td>sslp_5_25_50</td>
<td>w/o L-shaped</td>
<td>32</td>
<td>0.297</td>
<td>1.015</td>
<td>1.312</td>
</tr>
<tr>
<td>sslp_5_25_100</td>
<td>w/ L-shaped</td>
<td>17</td>
<td>0.124</td>
<td>2.236</td>
<td>2.36</td>
</tr>
<tr>
<td>sslp_5_25_100</td>
<td>w/o L-shaped</td>
<td>32</td>
<td>0.331</td>
<td>2.122</td>
<td>2.453</td>
</tr>
<tr>
<td>sslp_10_50_50</td>
<td>w/ L-shaped</td>
<td>298</td>
<td>171.339</td>
<td>663.667</td>
<td>835.006</td>
</tr>
<tr>
<td>sslp_10_50_50</td>
<td>w/o L-shaped</td>
<td>1024</td>
<td>1465.2</td>
<td>464.15</td>
<td>1929.35</td>
</tr>
<tr>
<td>sslp_10_50_100</td>
<td>w/ L-shaped</td>
<td>306</td>
<td>158.864</td>
<td>1215.43</td>
<td>1374.29</td>
</tr>
<tr>
<td>sslp_10_50_100</td>
<td>w/o L-shaped</td>
<td>1024</td>
<td>1320.62</td>
<td>831.894</td>
<td>2152.51</td>
</tr>
<tr>
<td>sslp_10_50_500</td>
<td>w/ L-shaped</td>
<td>180</td>
<td>10</td>
<td>3190</td>
<td>&gt;3200</td>
</tr>
<tr>
<td>sslp_10_50_500</td>
<td>w/o L-shaped</td>
<td>443</td>
<td>230.5</td>
<td>2969.5</td>
<td>&gt;3200</td>
</tr>
<tr>
<td>sslp_10_50_1000</td>
<td>w/ L-Shaped</td>
<td>68</td>
<td>2.5</td>
<td>3192.5</td>
<td>&gt;3200</td>
</tr>
<tr>
<td>sslp_10_50_1000</td>
<td>w/o L-shaped</td>
<td>363</td>
<td>116</td>
<td>3084</td>
<td>&gt;3200</td>
</tr>
<tr>
<td>sslp_15_45_5</td>
<td>w/ L-shaped</td>
<td>107</td>
<td>13.999</td>
<td>231.115</td>
<td>245.114</td>
</tr>
<tr>
<td>sslp_15_45_5</td>
<td>w/o L-shaped</td>
<td>392</td>
<td>2879</td>
<td>321</td>
<td>&gt;3200</td>
</tr>
<tr>
<td>sslp_15_45_10</td>
<td>w/ L-shaped</td>
<td>142</td>
<td>41</td>
<td>3159</td>
<td>&gt;3200</td>
</tr>
<tr>
<td>sslp_15_45_10</td>
<td>w/o L-shaped</td>
<td>374</td>
<td>2555</td>
<td>645</td>
<td>&gt;3200</td>
</tr>
</tbody>
</table>
The other important observation from running these two algorithms on the test case is the comparison of the plots of the upper and lower bounds as the algorithm iterates. As is shown in chart 1, the L2 algorithm tends to have a large gap between the upper and lower bounds up until the last iteration. This means that the algorithm will not be effective unless it is run to optimality, and if that takes too long, then the algorithm is not helpful at all. On the other hand, when the L-shaped cuts are added in, the algorithm gives much tighter upper and lower bounds in much less time. This means that even on the problems where the program was unable to find the solution in the time allotted, the best results that it found are still useful in that they are provably close to the optimal. The chart only gives one instance of the comparison between the upper and lower bounds, but this same pattern occurred in all of these test instances.
Adding in the L-shaped cuts can be expensive in computational time, especially if there are a lot of scenarios or the subproblems are large. Also, the initial L-shaped cuts are very effective at cutting down the feasible space of the master program, but as the cuts progress and the approximation of the subproblem value function becomes more exact, these cuts give less and less improvement in terms of cutting off feasible points in the master space. Avoiding solving the subproblems twice each iteration can be an effective way to improve the algorithm, so I tested the effect of only adding in the L-shaped cuts for a certain percentage of the overall iterations.

I ran the algorithm on the sslp_10_50_100 instance changing the number of L-shaped cuts that are allowed each time. The following table gives the number of cuts that I allowed, as well as the results of the algorithm.
<table>
<thead>
<tr>
<th>number</th>
<th>number/total enumeration</th>
<th>Iterations</th>
<th>Master time</th>
<th>Sub time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.03</td>
<td>1024</td>
<td>1332.87</td>
<td>835.457</td>
<td>2168.327</td>
</tr>
<tr>
<td>60</td>
<td>0.06</td>
<td>363</td>
<td>84.652</td>
<td>1466.5</td>
<td>1551.152</td>
</tr>
<tr>
<td>90</td>
<td>0.09</td>
<td>358</td>
<td>117.29</td>
<td>1316.99</td>
<td>1434.28</td>
</tr>
<tr>
<td>120</td>
<td>0.12</td>
<td>357</td>
<td>135.998</td>
<td>1308.17</td>
<td>1444.168</td>
</tr>
<tr>
<td>150</td>
<td>0.15</td>
<td>356</td>
<td>149.777</td>
<td>1260.73</td>
<td>1410.507</td>
</tr>
<tr>
<td>180</td>
<td>0.18</td>
<td>353</td>
<td>156.227</td>
<td>1289.47</td>
<td>1445.697</td>
</tr>
<tr>
<td>210</td>
<td>0.21</td>
<td>349</td>
<td>185.299</td>
<td>1271.06</td>
<td>1456.359</td>
</tr>
<tr>
<td>240</td>
<td>0.24</td>
<td>338</td>
<td>184.217</td>
<td>1239.81</td>
<td>1424.027</td>
</tr>
<tr>
<td>270</td>
<td>0.27</td>
<td>321</td>
<td>166.296</td>
<td>1215.53</td>
<td>1381.826</td>
</tr>
<tr>
<td>300</td>
<td>0.30</td>
<td>308</td>
<td>157.749</td>
<td>1220.55</td>
<td>1378.299</td>
</tr>
<tr>
<td>330</td>
<td>0.33</td>
<td>306</td>
<td>158.864</td>
<td>1215.43</td>
<td>1374.29</td>
</tr>
</tbody>
</table>

The table of results shows that the biggest gains are realized in the first few L-shaped cuts that are added. For this given instance, the amount of time that the problem took to solve decreases slightly as the number of cuts allowed is increased, but overall, it seems to be pretty flat. This implies that if the program is having trouble solving a certain instance, especially if the number of scenarios is high, or the master program is large, only adding the L-shaped cuts for the first few iterations may be an effective means of improving the run time.

V. Conclusions and Future Work

The main result of this project is that the L2 algorithm alone is not particularly good at solving mixed integer stochastic programs because computationally, it requires total enumeration of the first stage decision variables. This project gave one example of an improvement that can be made to algorithm to avoid this, adding non-supporting L-shaped typed cuts to the master program at each iteration. The cost of this is computational time in both the subproblems, to solve the relaxations, and in the master program, which grows twice as fast.
My computational experiments show that adding L-shaped cuts greatly improves the overall performance of the algorithm over the sslp set of test instances. The algorithm is especially improved for cases in which there is a large number of first stage variables that would take a long time to totally enumerate, and not too many subproblems that will take a long time to compute L-shaped cuts on, but I also found that in all the other cases, the new algorithm still gave significant improvements. I also showed the most of the improvements from the L-shaped cuts occur in the first few cuts made, so depending on the problem being optimized, it may be advantageous to stop adding the L-shaped cuts after a certain number of iterations.

A way to improve upon this algorithm would be to not solve all the mixed-integer subproblems to optimality during the early iterations. This would still give a decent approximation to their true value and might greatly improve the speed of the algorithm. Another idea for improving the performance of the L2 algorithm that would be useful to test is to see what the effect of a tighter overall lower bound for the cuts would be. The lower bound that I used was weak, and there are many ways to improve this bound such as by solving the relaxed stochastic program using the L-shaped method. Also, since the instances all come from the stochastic server location, it would be good to have another set of test cases to ensure that these results are not specific to this application. I think it would be important to directly compare the performance of the L2 algorithm with L-shaped cuts to the D2 algorithm of Higle and Sen. It would be interesting to see if the D2 algorithm is still significantly better.
VI. References


