Online Supplement to StAMPL: A Filtration-Oriented Modeling Tool for Multistage Stochastic Recourse Problems

Robert Fourer* Leo Lopes†

1 Examples

1.1 A Simple Financial Planning model from Birge and Louveaux (1997)

1.1.1 Model File

```
definestage 1;

set INSTR;
var Buy{INSTR} >= 0;

param initial_wealth;

subject to InvestAll:
    sum{i in INSTR} Buy[i] = initial_wealth;

############################################################
definestage 2..(stages()-1);

set INSTR;
var Buy{INSTR} >= 0;
```

*4er@iems.northwestern.edu
†leo@sie.arizona.edu (corresponding author)

```
16
17 param return{INSTR};
18
19 subject to ReinvestAll:
20     sum{i in INSTR} parent().Buy[i]*return[i] =
21     sum{i in INSTR} Buy[i];
22
23 ###############################################################
24 definestage stages();
25
26 set INSTR;
27 var Shortage >= 0;
28 var Overage >= 0;
29
30 param shortage_penalty;
31 param overage_reward;
32     check: shortage_penalty > overage_reward;
33 param return{INSTR};
34 param goal;
35
36 maximize FinalWealth:
37     overage_reward*Overage - shortage_penalty*Shortage;
38
39 subject to ReinvestAll:
40     sum{i in INSTR} parent().Buy[i]*return[i] +
41     Shortage - Overage = goal;
```

1.1.2 Data File

Note In the listing below, lines 12 and 19 are commented. These lines are not necessary for StAMPL, since the RandomElem class takes care of providing that information. However, when playing with the models individually within regular AMPL, it is convenient to have these lines in place.

```
1  setstages(4);
2
3  definestage 1;
4
5  set INSTR := STOCKS BONDS;
6  param initial_wealth := 55;
7
8  ####################################
9  definestage 2..(stages()-1);
10
11 set INSTR := STOCKS BONDS;
```

```
#param return:= STOCKS 1.25 BONDS 1.14;


####################################
definestage stages();

set INSTR := STOCKS BONDS;
#param return := STOCKS 1.25 BONDS 1.14;
param goal := 80;
param shortage_penalty := 4;
param overage_reward := 1;
```

1.1.3 Scenario Generation Code

```
#include <fstream>
#include <iostream>
#include <math.h>
#include <rvnode/rvnode.H>
#include <utils/leostl.H>

int main(void){

    const int nStages = 4;

    RandomElem retStocksGood, retStocksBad;
    RandomElem retBondsGood, retBondsBad;
    retStocksGood.param = retStocksBad.param = "return";
    retBondsGood.param = retBondsBad.param = "return";
    //push_back adds an element to the end of a list.
    retStocksGood.index.push_back("STOCKS");
    retStocksBad.index.push_back("STOCKS");
    retBondsGood.index.push_back("BONDS");
    retBondsBad.index.push_back("BONDS");
    retStocksGood.val = 1.25; retStocksBad.val = 1.06;
    retBondsGood.val = 1.14; retBondsBad.val = 1.12;

    int nScens = (int)pow(2,(nStages-1));
    const double pathprob = 1./nScens;

    int listSize = nScens / 2;
    RVNode *children = new RVNode[nScens];
    RVNode *parents;
    RVNode *parent, *goodChild, *badChild;

    for(int stage=nStages-1; stage > 0; stage-- ){
        parents = new RVNode[listSize];
        for(int i=0;i<listSize;i++){
```

```
            parent = parents+i;
            goodChild = children+2*i;
            badChild = goodChild+1;
            goodChild->stage = badChild->stage = stage+1;
            goodChild->pathprob = badChild->pathprob = pathprob;
            goodChild->elems.push_back(retStocksGood);
            goodChild->elems.push_back(retBondsGood);
            badChild->elems.push_back(retStocksBad);
            badChild->elems.push_back(retBondsBad);
            parent->children.push_back(*goodChild);
            parent->children.push_back(*badChild);
        }
        listSize /= 2;
        delete [] children;
        children = parents;
    }

    parent = parents;
    parent->stage = 1;
    parent->pathprob = pathprob;
    parent->NumberScens();
    parent->ReconcileProbabilities();
    cout << *parent << endl;
    delete [] parents;

}
```

1.2 A Foreign Currency Exchange model from Klaassen et al. (1990)

1.2.1 Model File

```
#####################################################################
definestage 1;

set OPTIONS;

var Buy{OPTIONS} >= 0;

param xchange;
param condexpxchange;
param volatility;
param us_r;
param fc_r := (xchange*(1 + us_r))/condexpxchange - 1;
param to_end := stages()-stage();
param temp1 := us_r - fc_r + (to_end)*((volatility^2)/2);
param temp2 := us_r - fc_r - (to_end)*((volatility^2)/2);
```

```
param temp3 := volatility*sqrt(to_end);
param c1{j in OPTIONS} :=
    CumNormal( (log(j/xchange) + temp1)/temp3 );
param c2{j in OPTIONS} :=
    CumNormal( (log(j/xchange) + temp2)/temp3 );
param price{j in OPTIONS} :=
    (1-c1[j])*(exp(-us_r*to_end)/j) -
    (1-c2[j])*(exp(-fc_r*to_end)/xchange);

minimize Cost:  sum{j in OPTIONS} Buy[j]*price[j];

subject to DoNotSpeculate:
    sum{j in OPTIONS} Buy[j] <= 1;

#################################################################
definestage 2..(stages()-1);

set OPTIONS;

var Buy{OPTIONS} >= 0;

param xchange;
param condexpxchange;
param volatility;
param us_r;
param fc_r := (xchange*(1 + us_r))/condexpxchange - 1;
param to_end := stages()-stage();
param temp1 := us_r - fc_r + (to_end)*((volatility^2)/2);
param temp2 := us_r - fc_r - (to_end)*((volatility^2)/2);
param temp3 := volatility*sqrt(to_end);
param c1{j in OPTIONS} :=
    CumNormal( (log(j/xchange) + temp1)/temp3 );
param c2{j in OPTIONS} :=
    CumNormal( (log(j/xchange) + temp2)/temp3 );
param price{j in OPTIONS} :=
    (1-c1[j])*(exp(-us_r*to_end)/j) -
    (1-c2[j])*(exp(-fc_r*to_end)/xchange);

minimize Cost:  (1/1+us_r)^(stage()-1) *
    sum{j in OPTIONS} Buy[j]*price[j];

#################################################################
definestage stages();

set OPTIONS;

param xchange;
param acceptable_xchange;
param us_r;

```

```
subject to OptionCost:
    xchange +
    sum{t in 1..(stages()-1)}(
        sum{j in OPTIONS}(
            parent(t).Buy[j]*(
                max(j-xchange,0) -
                (1+us_r)^(stages()-t)*parent(t).price[j]
            )
        )
    ) >= acceptable_xchange;

subject to DoNotSpeculate:
    sum{t in 1..(stages()-1)}
        sum{j in OPTIONS} parent(t).Buy[j]<=1;
```

1.2.2 Data File

```
setstages(5);

definestage 1..(stages()-1);

set OPTIONS := 0.44 0.50 0.57 0.63 0.70 0.76 0.83 0.89 0.96 1.02;

param xchange := .56;
param condexpxchange := .52;
param volatility := .11;
param us_r := .1;

definestage stages();

set OPTIONS := 0.44 0.50 0.57 0.63 0.70 0.76 0.83 0.89 0.96 1.02;

param xchange := .37;
param acceptable_xchange := .407;
param us_r := .1;
```

1.2.3 Example Scenario Tree generation code

This example exploits the fact that a ternary tree has $3^{(t-1)}$ nodes at stage t, and ternary lattices have the following properties:

- the number of nodes at stage t of a ternary tree is the tth odd number. Thus, since $\sum_{i=1}^{k}(2i-1) = k^2$, if all the nodes are put in a sequence, the index of the last element

of the sequence at stage t is t^2. This provides a convenient mechanism for storing the instantiations of the random variables in the lattice in a one-dimensional array.

- the kth node at stage t is always connected to the kth, $(k+1)$st, and $(k+2)$nd nodes at stage $t+1$.

These two facts allow us to build a ternary tree from a ternary lattice by copying children trees onto their parents' child lists, from the last stage to the first:

- **Given**:
  - a stage t
  - a list c of length $2t+1$ of subtrees (these will be the children).
- Create a list p of length $2t-1$ of nodes (these will be the parents)
- For each parent p_i, *copy* subtrees c_i, c_{i+1}, and c_{i+2} onto the child list of p_i.

At the next iteration $(t-1)$, use the newly generated list of parents p as the list of children c. The implementation of this algorithm follows:

```
#include <fstream>
#include <iostream>
#include <math.h>
#include <rvnode/rvnode.H>
#include <utils/leostl.H>

int main(void){
    const int stages = 5;
    const int arity = 3;
    const double accxchg[9] =
        {.407,.416,.423,.429,.444,.466,.494,.527,.564};
    const double xchg[25] = {
        .56,
        .47,.52,.57,
        .43,.46,.49,.55,.61,
        .39,.42,.44,.49,.52,.58,.65,
        .37,.39,.41,.43,.46,.50,.55,.61,.68
    };
    RandomElem rexchg;
    rexchg.param = "xchange";
```

```
21      RandomElem reaccxchg;
22      reaccxchg.param = "acceptable_xchange";
23
24      RVNode *children = new RVNode[(stages-1)*2+1];
25      RVNode *rn;
26      const double pathprob = 1./pow(arity,stages-1);
27      for(int stage=stages-1;stage>0;stage--){
28          const int card = stage*2+1, zeroidx = stage*stage;
29          rn = children;
30          const double condprob = 1./pow(arity,stage);
31          for(int i=0;i<card;i++){
32              rexchg.val = xchg[zeroidx+i];
33              rn->prob = condprob;
34              rn->pathprob = pathprob;
35              rn->stage = stage+1;
36              rn->elems.push_back(rexchg);
37              if(stage == stages-1){
38                  reaccxchg.val = accxchg[i];
39                  rn->elems.push_back(reaccxchg);
40              }
41              rn++;
42          }
43          RVNode *parents = new RVNode[card-2];
44          rn = parents;
45          RandomElem cex;
46          cex.param = "condexpxchange";
47          for(int i=0;i<card-2;i++){
48              cex.val = 0;
49              for(int j=0;j<arity;j++){
50                  cex.val += xchg[zeroidx+i+j]/arity;
51                  rn->children.push_back(children[i+j]);
52              }
53              rn->elems.push_back(cex);
54              rn++;
55          }
56          delete [] children;
57          children = parents;
58      }
59
60      rn--;
61      rn->stage = 1;
62      rn->prob = 1;
63      rn->pathprob = 1./81;
64      rn->NumberScens();
65
66      cout << *rn << endl;
67      delete [] rn;
68
69  }
```

References

Birge, J. R., F. Louveaux, 1997. *Introduction to Stochastic Programming.* Springer-Verlag, New York.

Klaassen, P., J. F. Shapiro, D. E. Spitz, 1990. Sequential decision models for selecting currency options. Technical Report 133-90, MIT, International Financial Services Research Center.