FLOPC++

An algebraic modeling language embedded in C++

OR 2006, Karlsruhe

6.-8. September 2006

-

Tim Hultberg

tim.hultberg@eumetsat.int

EUMETSAT, Darmstadt

FLOPC++

Formulation of Linear Optimization Problems in C++

- Algebraic modelling language.
- Implemented as a C++ class library.
- Declarative optimization modelling (similar to GAMS, AMPL and AIMMS), within a C++ program.
- LP's and MIP's
- Open source (Common Public License version 1.0)
- https://projects.coin-or.org/FlopC++
- New users are appreciated.

Overview of the talk

1. Motivation
2. How to use FLOPC++ (tips and tricks)
3. Implementation
4. Future plans (Stochastic Programming extensions)
5. Conclusion

Traditional AMLs

Strengths …

- Syntax close to the notation used by most modellers
- Instance generation is automated
- Easy to modify models
- Representation is readable by both humans and computers

Weaknesses …

- Hard to integrate with other software components
- Limited procedural support for algorithm development
- Limited model/program structuring facilities
- Limited flexibility and extendibility

Embedding an opt. model ...

in a C/C++ application.

- **1. Using the callable library of the solver.**

```
...
// code to generate a problem instance
// in the format of the solver

lp.copylpdata(n, m, minimize, c, b, sense,
              Cst, Clg, Rnr, Elm, l, u);

solstat = lp.primopt();

if (solstat==CPX_OPTIMAL) lp.getX(solution);
...
```

Embedding an opt. model ...

in a C/C++ application.

- **2. Using an algebraic modelling language.**
  1. Write code to generate a data file.
  2. Write code to spawn the algebraic modelling language interpreter.
  3. Write code to parse the results from a file.

Declarative + procedural

- column generation
- decomposition
- heuristics
- etc.

We must combine declarative and procedural modelling.

Two approaches:

1. Add procedural constructs to an algebraic modelling language.
2. Add declarative modelling constructs to a general purpose programming language.

Other modelling libraries

'MP' (Søren S. Nielsen) C++

Planner (ILOG) C++

Concert Technology (ILOG) C++

LP Toolkit (Euro-decision) C++

XBSL (Dash) C

AMMO (Drayton Analytics) COM

Component Libraries (ILOG) COM, based on OPL

OptiMax2000 (Maximal Software) COM, based on MPL

EMOSL (Dash) C, based on XPRESS-MP

Comparison I

ILOG Concert Technology

```
for (IloInt s=0; s<numS; s++)
    model.add( IloSum(x[s]) <=  capacity[s] );
```

FLOPC++

```
supply(S) = sum(D, x(S,D)) <= capacity(S);
```

Comparison II

ILOG Concert Technology

```
for (IloInt d=0; d<numD; d++) {
    IloExpr total(env);
    for (IloInt s=0; s<numS; s++) {
        total += x[s][d];
    }
    model.add( total >= dem[d] );
    total.end();
}
```

FLOPC++

```
demand(D) = sum(S, x(S,D)) >= dem(D);
```

Non-Commercial

From the web pages of Karlsruhe Universität:

Universität Karlsruhe (TH) - Institut für
Wirtschaftstheorie und Operations Research

WiOR

Home / Programmbibliothek OR / Modeling Languages / Non Commercial /

Modeling Languages - Non-Commercial Software

Unfortunately we currently have no packages in the list.

An example

```
main() {
  MP_set S(numS),D(numD);
  MP_subset<2> Link(S,D);
  MP_data SUPPLY(S), DEMAND(D), d(Link), c(Link);
  MP_variable  x(S,D);
  MP_constraint supply(S), demand(D);

  c(Link) = f * d(Link) / 1000;

  supply(S) =  sum( Link(S,D), x(Link) ) <= SUPPLY(S);
  demand(D) =  sum( Link(S,D), x(Link) ) >= DEMAND(D);

  minimize( sum(Link, c(Link) * x(Link)) );

  x.display("Optimal transportation plan");
}
```

MP_expression

- Explicitly name a linear expression without adding rows or columns to the generated instance.
- Alternative to accounting variables

```
MP_expression production(Products);

production(p) = sum(T(t), xi(p,t) + xo(p,t));

Min_prod(p) = production(p) >= MIN_PROD(p);

E_limit(q) =
    sum(Products(p), E(p,q)*production(p)) <= LIMIT(q);
```

minimize_max

- Higher level modelling extensions
- Modelling objects as parameters

```
void MP_model::minimize_max(MP_set &s, const MP_expression  &obj) {
     MP_variable v;
     MP_constraint c(s);
     add(c);
     c(s) = v() >= obj;
     minimize(v());
}
....
minimize_max( J, s(J)+D(J) );
```

Problem modification I

Routines to change

- the objective function
- lower and upper bounds on variables
- right hand side and sense of constraints
- variable types

or add or delete

- columns (variables)
- rows (constraints)

without regeneration of the problem.

Problem modification II

```
for (int j=0; j<numDistributionCenters; j++)
  subproblem->setRowBounds(selling(j), received.level(j), received.level(j
double objsub = 0.0;
for (int s=0; s<numScenarios; s++) {
  for (int j=0; j<numDistributionCenters; j++)
    subproblem->setRowBounds( selmax(j), Demand[s][j], Demand[s][j]);
  subproblem->resolve();
  objsub += prob[s]*subproblem->getObjValue();
  for (int j=0; j<numDistributionCenters; j++) {
    cutCnst += prob[s]*selmax.price(j)*Demand[j][s];
    coeff(j) += prob[s]*selling.price(j);
  }
}
upperbound = min(upperbound, objmaster + objsub);
if ((upperbound-lowerbound) < 0.01*(1+fabs(lowerbound))) break;
masterproblem.addRow( theta() >= cutCnst + sum(j, coeff(j)*received(j)) );
masterproblem->resolve();
lowerbound = masterproblem->getObjValue();
objmaster = lowerbound - theta.level();
```

(full model in stochBenders.cpp)

Cyclic index sets

```
enum {t12pm_6am, t6am_9am, t9am_3pm, t3pm_6pm, t6pm_12pm, numT};

MP_set T(numT); T.cyclic();
MP_set G(numG);

MP_integer_variable  n(G,T);  // number of generators in use
MP_variable          s(G,T);  // number of generators started up

MP_constraint        st(G,T); // start_up definition

st(G,T) =   s(G,T) >= n(G,T) - n(G,T-1);
```

(full model in magic.cpp)

Implementation

- keep track of correspondence between individual variables (/constraints) and its column (/row) index in the generated problem instance
- expressions evaluate to abstract syntax trees

```
supply(S) =  sum( Link(S,D), x(Link) ) <= SUPPLY(S);
demand(D) =  sum( Link(S,D), x(Link) ) >= DEMAND(D);
```

- generation takes place in the leaf nodes, information from upper level nodes is passed recursively down in the tree
- expressions are reference counted to avoid memory leaks

Miscellaneous

- Selfcontained.
  - Required Coin-OR projects (CoinUtils, Cgl, Clp, Cbc, Osi, MSVisualStudio, BuildTools) included.
- Solver independent.
  - Uses OSI (Optimization Solver Interface) from Coin-OR (Clp, Cbc, CPLEX, DyLP, FortMP, GLPK, MOSEK, OSL, SoPlex, SYMPHONY, Vol, XPRESS-MP)
- Windows / MS VisualStudio supported.

Documentation

1. Doxygen Documentation. (Philip Walton)
2. 'User Documentation' (Christian Woellenstein)
3. Small FAQ (pose questions to the mailing list)
4. Examples (currently 18 different)

Open source

Initiation of optimization projects:

- One less tool to buy/license

Long term maintenance:

- Compiler release versioning – do we have the right run time library for this platform?
- Operating system release library versions – do we have the build for this machine?
- Licensing hassles (revisit the code because of licensing changes). Must keep feeding the maintenance cost.

Development:

- Users are able to contribute - and they do.

Plans

- Symbolic model checks

```
MP_set I(5), J(2);
MP_variable x(I,J);
MP_constraint c;
c() =  sum(I*J, x(J,I)) <= 100
```

  generates this (without warnings):

```
x(0,0) + x(0,1) + x(1,0) + x(1,1) <= 100
```

- Non-linear programming modelling, etc. Possible, but...
- Stochastic programming modelling.
  - In the pipeline...

SP extensions

SP with current FLOPC++

- Benders decomposition (stochBenders.cpp)
- Recursive formulation (stampl.cpp)

Work in progres (with Alan King)

- `MP_stage T(numStages);`
- `MP_stochastic_data Return(T,INSTR);`
- Uses Coin-Smi (Stochastic Modelling interface)
- Generate core model
- Add scenarios

SP example

```
MP_set INSTR(numINSTR);
MP_stage T(numStages);
MP_stochastic_data Return(T,INSTR);

MP_variable Buy(T,INSTR), Shortage, Overage;
MP_constraint InvestAll, ReinvestAll(T), Goal;

InvestAll() = sum(INSTR, Buy(0,INSTR)) == initial_wealth;

ReinvestAll(T+1) = sum(INSTR, Buy(T,INSTR) * Return(T,INSTR))
                     == sum(INSTR, Buy(T+1,INSTR));

Goal() = sum(INSTR, Buy(T.last(),INSTR) * Return(T.last(),INSTR))
           == goal - Shortage() + Overage();

maximize( Overage() - 4*Shortage() );
```

Conclusion

Advantages in addition to traditional modelling languages.

- Lightweight
- Open source
- Multi paradigm
- Fast problem generation
- Seamless integration with applications
- Efficient and powerful for customized algorithms (decomposition, column generation, cut generation)
- Modelling objects are first class C++ types. This allows higher level modelling extensions (such as the minimize max objective) to be implemented.

Disadvantages ?