FLOPC++
An Algebraic Modeling Language Embedded in C++

Tim Helge Hultberg

EUMETSAT, Darmstadt, Germany
tim.hultberg@eumetsat.int

Summary. FLOPC++ is an open source algebraic modeling language implemented as a C++ class library. It allows linear optimization problems to be modeled in a declarative style, similar to algebraic modeling languages, such as GAMS and AMPL, within a C++ program. The project is part of COmputational INfrastructure for Operations Research (COIN-OR) and uses its Open Solver Interface (OSI) to achieve solver independence.

1 Introduction

Algebraic modeling languages are important tools in Operations Research, their benefits are well known [1]: They allow a model representation which is readable by both humans and computers, using a syntax close to the notation used by most modelers. They automate the generation of model instances, freeing the modeler from the burden of generating the low level matrix format needed by the solver. In summary, they make implementation and maintenence of optimization models much easier.

However, traditional algebraic modeling languages do also have major weaknesses:

- Limited flexibility for integration with other software components
- Limited procedural support for algorithm development
- Limited model/program structuring facilities
- Limited extendibility

In order to mitigate these weaknesses, most algebraic modeling languages have incorporated procedural constructs, which allow the construction of algorithms around the optimization models. It has also become common to provide some sort of support for embedding the models in applications.

A natural alternative to extending the existing modeling languages is to implement modeling abstractions within an existing general-purpose object-oriented programming language, C++, in order to enable the "Formulation of Linear Optimization Problems in C++". This is what FLOPC++ is all about.

The work on FLOPC++ began in year 2000 when the author had developed a production model for a Danish poultry butchery in GAMS and was faced with the practical problems of integrating the model in a GUI for collecting data and displaying the results. Discussing these problems with Søren Nielsen, he pointed me to his excellent paper [3], which was the main inspiration for the start of the development of FLOPC++. [2]

The first versions of FLOPC++ used the CPLEX callable library as its solver, but the need for supporting several independent solvers soon became apparent. Luckily, the simultaneous appearance of the Osi (Open Solver Interface) providing an abstract base class to a generic linear programming (LP) solver, along with derived classes for specific solvers, made this an easy task.

The algebraic representation of a model convey structural information that is hidden in the matrix representation required by the solver. Sometimes this information can be used to develop specialized solution algorithms using a general linear optimization solver for subproblems. Examples of such algorithms include decomposition, column generation and model specific cutting plane algorithms. The development of such algorithms is greatly facilitated by working in an environment where the algebraic representation of the model is available. This kind of algorithms can often be implemented in traditional modeling languages, but the implementations are mostly hopelessly inefficient because problems are passed to the solver via file interfaces and most importantly because slightly modified problems get regenerated from scratch. FLOPC++ offers the possibility to modify problem instances and warm start the solver from the last solution.

In Section 2 we give a brief introduction to the implementation of FLOPC++ without going into every aspect, before the conclusion in Section 3.

2 Implementation

Consider the following excerpt from the FLOPC++ model "transport.cpp" (the full model is one of the examples in the model library which comes with the distribution):

```
MP_set S(numS);          // Sources
MP_set D(numD);          // Destinations
MP_subset<2> Link(S,D);  // Transp. links (subset of S*D)
MP_data DEMAND(D);
MP_data SUPPLY(S);
MP_variable x(Link);
MP_constraint supply(S);
MP_constraint demand(D);

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

We see that C++ classes are available to represent the basic algebraic modeling constructs: index sets, parameters, variables and constraints. There is also a class for representing dummy indices (MP_index), but since index sets can also be used as dummy indices, they are not needed for this particular model.