A Gentle Introduction to COIN-OR's Optimization Solver Interface (OSI)

Resources and Examples
CORS/INFORMS Banff 2004

June 2, 2004

Note: This document is intended as a reference aid. The material is not presented in the same order as the presentation.

1 Web resources

- COIN-OR website: `www.coin-or.org`
- COIN-OR tutorials site: `http://sagan.ie.lehigh.edu/coin/`
- C++ Annotations by Frank B. Brokken; intended for people who know C and want to learn C++. `http://www.icce.rug.nl/documents/cplusplus/cplusplus.html`
- C/C++ Reference, `http://www.cppreference.com/`

2 Getting help with OSI

We want to help make your use of OSI successful!

- First review the appropriate documentation—the answer may be there.
- Send email to `coin-discuss@www-124.ibm.com`. **This address is likely to change soon–check www.coin-or.org before sending.**
- In your email, give as much detail as you can:
  - Operating system
  - COIN-OR modules (OSI, CLP, etc.)
  - Solvers
  - Error messages

3 Optimization Solver Interface (OSI)

Uniform interface to LP/IP solvers:

- CLP (COIN-OR)
- CPLEX (ILOG)
- DyLP (BonsaiG LP Solver)
- GLPK (GNU LP Kit)
- OSL (IBM)
- SoPlex (Konrad-Zuse-Zentrum für Informationstechnik Berlin)
- Volume (COIN-OR)
- XPRESS (Dash Optimization)
- Mosek interface is written and will enter the repository soon

4 Procedures

4.1 Steps to prepare OSI

1. Download source code
2. Configure based on available solvers
3. Compile
4. Create a makefile for your project (optional)
5. Use OSI in your code

4.2 Downloading, configuring, and compiling OSI

- Download tarball from www.coin-or.org: `Osi_2003Oct17.tgz`. You may also want `Osi-doc_2003Oct17.tgz`. (Replace `Oct17` with a current date when you acquire the code.)
- Repository can also be accessed with CVS.
- Configuration in the Makefiles directory
  - Edit `Makefile.location` to tell OSI which solvers are available and where they are
  - Edit `Makefile.<platform>` (e.g. `Makefile.Linux`, `Makefile.SunOS`) if you want to control the compiler, linker, etc. The default settings are probably OK.

- Compile with the command `make` in the directory Coin and then Osi. May need to do `make` in subdirectories of Osi as well, such as OsiGlpk and OsiDylp, depending on the solvers available.
- Create a Makefile for your project that indicates the location of OSI headers and libraries. An example is given later.

4.3 Using OSI in your code

- Solver dependent parts:
  - Include the header files for solver(s) you want to use.
  - Create an OsiXxxSolverInterface object.
- Solver independent:
  - Call functions to load/create a problem.
  - Call functions to solve the problem.
  - Call functions to report on the solution, modify the problem and re-solve, or do something else

5 Examples

5.1 basic.cpp

```
// Bare bones example of using COIN-OR OSI

#include <iostream>
#include "OsiClpSolverInterface.hpp"

int
main(void)
{
  // Create a problem pointer.  We use the base class here.
  OsiSolverInterface *si;

  // When we instantiate the object, we need a specific derived class.
  si = new OsiClpSolverInterface;

  // Read in an mps file.  This one's from the MIPLIB library.
  si->readMps("p0033");

  // Solve the (relaxation of the) problem
  si->initialSolve();

  // Check the solution
  if ( si->isProvenOptimal() )
    {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      int n = si->getNumCols();
      const double *solution;
      solution = si->getColSolution();
      // We could then print the solution or examine it.
    }
  else
    {
      stdd::cout << "Didn't find optimal solution." << std::endl;
      // Could then check other status functions.
    }

  return 0;
}
```

5.2 basic2.cpp

```
// Bare bones example of using COIN-OR OSI

#include <iostream>
//#include "OsiClpSolverInterface.hpp"
#include "OsiGlpkSolverInterface.hpp"

int
main(void)
{
  // Create a problem pointer.  We use the base class here.
  OsiSolverInterface *si;

  // When we instantiate the object, we need a specific derived class.
  //si = new OsiClpSolverInterface;
  si = new OsiGlpkSolverInterface;

  // Read in an mps file.  This one's from the MIPLIB library.
  si->readMps("p0033");

  // Solve the (relaxation of the) problem
  si->initialSolve();

  // Check the solution
  if ( si->isProvenOptimal() )
    {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      int n = si->getNumCols();
      const double *solution;
      solution = si->getColSolution();
      // We could then print the solution or examine it.
    }
  else
    {
      std::cout << "Didn't find optimal solution." << std::endl;
      // Could then check other status functions.
    }

  return 0;
}
```

5.3 query.cpp

```
// Example of using COIN-OR OSI
// Demonstrates some problem and solution query methods

#include <iostream>
#include "OsiClpSolverInterface.hpp"

int
main(void)
{
  // Create a problem pointer.  We use the base class here.
  OsiSolverInterface *si;

  // When we instantiate the object, we need a specific derived class.
  si = new OsiClpSolverInterface;

  // Read in an mps file.  This one's from the MIPLIB library.
  si->readMps("p0033");

  // Display some information about the instance
  int nrows = si->getNumRows();
  int ncols = si->getNumCols();
  int nelem = si->getNumElements();
  std::cout << "This problem has " << nrows << " rows, "
       << ncols << " columns, and " << nelem << " nonzeros." << std::endl;

  double const * upper_bounds = si->getColUpper();
  std::cout << "The upper bound on the first column is " << upper_bounds[0]
       << std::endl;
  // All the information about the instance is available with similar methods

  // Solve the (relaxation of the) problem
  si->initialSolve();

  // Check the solution
  if ( si->isProvenOptimal() )
    {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      // Examine solution
      int n = si->getNumCols();
      const double *solution;
      solution = si->getColSolution();

      std::cout << "Solution: ";
      for (int i = 0; i < n; i++)
        std::cout << solution[i] << " ";
      std::cout << std::endl;

      std::cout << "It took " << si->getIterationCount() << " iterations"
           << " to solve." << std::endl;
    }
  else
    {
      std::cout << "Didn't find optimal solution." << std::endl;

      // Check other status functions.  What happened?
      if (si->isProvenPrimalInfeasible())
        std::cout << "Problem is proven to be infeasible." << std::endl;
      if (si->isProvenDualInfeasible())
        std::cout << "Problem is proven dual infeasible." << std::endl;
      if (si->isIterationLimitReached())
        std::cout << "Reached iteration limit." << std::endl;
    }
  return 0;
}
```

5.4 parameters.cpp

```
// Example of using COIN-OR OSI
// Demonstrates some problem and solution query methods
// Also demonstrates some parameter setting

#include <iostream>
#include "OsiClpSolverInterface.hpp"

int
main(void)
{
  // Create a problem pointer.  We use the base class here.
  OsiSolverInterface *si;

  // When we instantiate the object, we need a specific derived class.
  si = new OsiClpSolverInterface;

  // Read in an mps file.  This one's from the MIPLIB library.
  si->readMps("p0033");

  // Display some information about the instance
  int nrows = si->getNumRows();
  int ncols = si->getNumCols();
  int nelem = si->getNumElements();
  std::cout << "This problem has " << nrows << " rows, "
       << ncols << " columns, and " << nelem << " nonzeros." << std::endl;

  double const * upper_bounds = si->getColUpper();
  std::cout << "The upper bound on the first column is " << upper_bounds[0]
       << std::endl;
  // All the information about the instance is available with similar methods


  // Before solving, indicate some parameters
  si->setIntParam( OsiMaxNumIteration, 10);
  si->setDblParam( OsiPrimalTolerance, 0.001 );

  // Can also read parameters
  string solver;
  si->getStrParam( OsiSolverName, solver );
  std::cout << "About to solve with: " << solver << std::endl;


  // Solve the (relaxation of the) problem
  si->initialSolve();

  // Check the solution
  if ( si->isProvenOptimal() )
    {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      // Examine solution
      int n = si->getNumCols();
      const double *solution;
      solution = si->getColSolution();

      std::cout << "Solution: ";
      for (int i = 0; i < n; i++)
        std::cout << solution[i] << " ";
      std::cout << std::endl;

      std::cout << "It took " << si->getIterationCount() << " iterations"
           << " to solve." << std::endl;
    }
  else
    {
      std::cout << "Didn’t find optimal solution." << std::endl;

      // Check other status functions.  What happened?
      if (si->isProvenPrimalInfeasible())
        std::cout << "Problem is proven to be infeasible." << std::endl;
      if (si->isProvenDualInfeasible())
        std::cout << "Problem is proven dual infeasible." << std::endl;
      if (si->isIterationLimitReached())
        std::cout << "Reached iteration limit." << std::endl;
    }

  return 0;
}
```

5.5 build.cpp

```
// Example of using COIN-OR OSI, building the instance internally
// with sparse matrix object

#include <iostream>
#include "OsiClpSolverInterface.hpp"
#include "CoinPackedMatrix.hpp"
#include "CoinPackedVector.hpp"

int
main(void)
{
  // Create a problem pointer.  We use the base class here.
  OsiSolverInterface *si;

  // When we instantiate the object, we need a specific derived class.
  si = new OsiClpSolverInterface;

  // Build our own instance from scratch

  /*
   * This section adapted from Matt Galati's example
   * on the COIN-OR Tutorial website.
   *
   * Problem from Bertsimas, Tsitsiklis page 21
   *
   *  optimal solution: x* = (1,1)
   *
   *  minimize -1 x0 - 1 x1
   *  s.t       1 x0 + 2 x1 <= 3
   *            2 x0 + 1 x1 <= 3
   *              x0        >= 0
   *              x1        >= 0
   */

  int n_cols = 2;
  double * objective    = new double[n_cols];//the objective coefficients
  double * col_lb       = new double[n_cols];//the column lower bounds
  double * col_ub       = new double[n_cols];//the column upper bounds

  //Define the objective coefficients.
  //minimize -1 x0 - 1 x1
  objective[0] = -1.0;
  objective[1] = -1.0;

  //Define the variable lower/upper bounds.
  // x0 >= 0   =>  0 <= x0 <= infinity
  // x1 >= 0   =>  0 <= x1 <= infinity
  col_lb[0] = 0.0;
  col_lb[1] = 0.0;
  col_ub[0] = si->getInfinity();
  col_ub[1] = si->getInfinity();

  int n_rows = 2;
  double * row_lb = new double[n_rows]; //the row lower bounds
  double * row_ub = new double[n_rows]; //the row upper bounds

  //Define the constraint matrix.
  CoinPackedMatrix * matrix =  new CoinPackedMatrix(false,0,0);
  matrix->setDimensions(0, n_cols);

  //1 x0 + 2 x1 <= 3  =>  -infinity <= 1 x0 + 2 x2 <= 3
  CoinPackedVector row1;
  row1.insert(0, 1.0);
  row1.insert(1, 2.0);
  row_lb[0] = -1.0 * si->getInfinity();
  row_ub[0] = 3.0;
  matrix->appendRow(row1);

  //2 x0 + 1 x1 <= 3  =>  -infinity <= 2 x0 + 1 x1 <= 3
  CoinPackedVector row2;
  row2.insert(0, 2.0);
  row2.insert(1, 1.0);
  row_lb[1] = -1.0 * si->getInfinity();
  row_ub[1] = 3.0;
  matrix->appendRow(row2);

  //load the problem to OSI
  si->loadProblem(*matrix, col_lb, col_ub, objective, row_lb, row_ub);

  //write the MPS file to a file called example.mps
  si->writeMps("example");



  // Solve the (relaxation of the) problem
  si->initialSolve();

  // Check the solution
  if ( si->isProvenOptimal() )
    {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      int n = si->getNumCols();
      const double *solution;
      solution = si->getColSolution();
      // We could then print the solution or examine it.
    }
  else
    {
      std::cout << "Didn’t find optimal solution." << std::endl;
      // Could then check other status functions.
    }

  return 0;
}
```

5.6 specific.cpp

```
// Example of using COIN-OR OSI
// including accessing solver-specific functions

#include <iostream>
#include "OsiClpSolverInterface.hpp"

int
main(void)
{
  // Create a problem pointer.  We use the base class here.
  OsiSolverInterface *si;

  // When we instantiate the object, we need a specific derived class.
  si = new OsiClpSolverInterface;

  // The next few lines are solver-dependent!
  ClpSimplex * clpPointer;
  clpPointer = (dynamic_cast<OsiClpSolverInterface *>(si))->getModelPtr();

  clpPointer->setLogLevel(0);
  //clpPointer->setMaximumIterations(10);
  // Could tell Clp many other things

  // Read in an mps file.  This one's from the MIPLIB library.
  si->readMps("p0033");

  // Solve the (relaxation of the) problem
  si->initialSolve();

  // Check the solution
  if ( si->isProvenOptimal() )
    {
      std::cout << "Found optimal solution!" << std::endl;
      std::cout << "Objective value is " << si->getObjValue() << std::endl;

      int n = si->getNumCols();
      const double *solution;
      solution = si->getColSolution();
      // We could then print the solution or examine it.
    }
  else
    {
      std::cout << "Didn't find optimal solution." << std::endl;
      // Could then check other status functions.
    }

  return 0;
}
```

5.7 Features of OSI not demonstrated by the examples

- Several methods for loading problems
- Re-solve after modifying problem
- Integer programs
- "Hints" for presolving, scaling, using dual simplex
- Warm starts and hot starts
- Simplex-level controls for basis, pivots, etc. (currently only implemented for CLP, I think)

6 Example Makefile

```
CXX := g++

COIN_DIR := $(HOME)/research/computation/COIN
GLPK_DIR := $(HOME)/research/computation/glpk-4.1

COIN_INC_DIR := $(COIN_DIR)/include
GLPK_INC_DIR := $(GLPK_DIR)/include

CXX_FLAGS := -I$(COIN_INC_DIR) -I$(GLPK_INC_DIR)

COIN_LIB_DIR := $(COIN_DIR)/lib
GLPK_LIB_DIR := $(GLPK_DIR)

LD_FLAGS := -L$(COIN_LIB_DIR) -L$(GLPK_LIB_DIR)
LD_FLAGS += -Wl,-R,$(COIN_LIB_DIR):$(GLPK_LIB_DIR)

LIB_FLAGS :=  -lCoin -lOsi -lOsiGlpk -lOsiClp -lClp -lglpk -lm


default: basic

%.o: %.cpp
        $(CXX) $(CXX_FLAGS) -c $<

basic: basic.o
        $(CXX) -o $@ $(CXX_FLAGS) $(LD_FLAGS) $< $(LIB_FLAGS)
```