Optimization Services 1.0 User's Manual

Robert Fourer, Jun Ma, Kipp Martin

June 10, 2007

Abstract

This is the User's Manual for the Optimization Services (OS) project. The objective of (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site for more information.

Contents

List of Figures

List of Tables

1 Introduction

The objective of Optimization Services (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site for more information.

2 Download and Installation

OS is released as open source code under the Common Public License (CPL). This project was created by Robert Fourer, Jun Ma, and Kipp Martin. The code has been written primarily by Jun Ma, Kipp Martin, Robert Fourer and Huanyuan Sheng, the first two are the COIN project leaders for OS. Below we describe different methods for obtaining the C++ source code and binaries.

2.1 Obtaining the Source Code Subversion Repository (SVN)

The C++ source code can be obtained using Subversion. Users with Unix operating systems will most likely have an svn client. For Windows users wishing to obtain and SVN client we recommend ***kipp fillin

The OS project page with a Wiki is available at `projects.coin-or.org\OS`. Execute the following steps to get the source code using SVN.

Step 1: Connect to a directory where you want the OS project to go. The following command will download the project into the directory COIN-OS

```
svn co https://projects.coin-or.org/svn/OS/stable/1.0 COIN-OS
```

Step 2: Connect to the distribution root directory.

```
cd COIN-OS
```

Step 3: Run the configure script that will generate the makefiles.

```
./configure
```

Step 4: Run the make files.

```
make
```

Step 5: Run the unitTest.

```
make test
```

Depending upon which third party software you have installed, the result of running the unitTest should look something like:

```
HERE ARE THE UNIT TEST RESULTS:

Solved problem avion2.osil with Ipopt
Solved problem HS071.osil with Ipopt
Solved problem rosenbrockmod.osil with Ipopt
Solved problem parincQuadratic.osil with Ipopt
Solved problem parincLinear.osil with Ipopt
Solved problem callBack.osil with Ipopt
Solved problem callBackRowMajor.osil with Ipopt
Solved problem parincLinear.osil with Clp
Solved problem p0033.osil with Cbc
Solved problem rosenbrockmod.osil with Knitro
Solved problem callBackTest.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem p0033.osil with SYMPHONY
Solved problem parincLinear.osil with DyLP
Solved problem lindoapiaddins.osil with Lindo
Solved problem rosenbrockmod.osil with Lindo
Solved problem parincQuadratic.osil with Lindo
Solved problem wayneQuadratic.osil with Lindo
Test the MPS -> OSiL converter on parinc.mps usig Cbc
Test the AMPL nl -> OSiL converter on hs71.nl using LINDO
Test a problem written in b64 and then converted to OSInstance
Successful test of OSiL parser on problem parincLinear.osil
Successful test of OSrL parser on problem parincLinear.osrl
Successful test of prefix and postfix conversion routines on problem rosenbrockmod.osil
Successful test of all of the nonlinear operators on file testOperators.osil
Successful test of AD gradient and Hessian calculations on problem CppADTestLag.osil


CONGRATULATIONS! YOU PASSED THE UNIT TEST
```

If you do not see

```
CONGRATULATIONS! YOU PASSED THE UNIT TEST
```

then you have not passed the unitTest and hopefully some semi-inteligble error message was given. CONGRATULATIONS! YOUPASSEDD THE UNIT TEST

Step 6: Install the libraries.

```
make install
```

This will install all of the libraries in the `lib` directory under the distribution root.

The above steps are fully tested on Mac/Unix, Linux, and on Windows using either MINGW/MSYS or CYGWIN. Popular compilers like gcc/g++ or windows native compiler cl.exe can all be used.

Note if you download the OS package, you get these additional COIN-OR projects.

- Cbc `projects.coin-or.org\Cbc`
- Clp `projects.coin-or.org\Clp`
- CppAD `projects.coin-or.org\CppAD`
- Dylp `projects.coin-or.org\Dylp`
- Osi `projects.coin-or.org\Osi`
- SYMPHONY `projects.coin-or.org\SYMPHONY`

2.2 Obtaining the Source Code From a Tarball

2.3 Obtaining a Visual Studio Project

2.4 Obtaining the Binaries

2.5 Platforms

	Mac	Linux	Cyg-gcc	Msys-cl	Msys-gcc	MSVS
AMPL-Client	x	x		x		
Cbc	x	x	x	x		
Clp	x	x	x	x		
Cplex	x	x				
DyLP	x	x	x	x		
Ipopt	x	x				
Knitro	x					
Lindo	x	x		x		
SYMPHONY	x	x	x	x		

Platform Detail:

	Operating System	**Compiler**	**Hardware**
Mac	Mac OS X 10.4.9	gcc 4.0.1	Power PC
Linux	Red Hat 3.4.6-8	gcc 3.4.6	Dell Intel 32 bit chip
Cyg-gcc	Windows 2003 Server	gcc 3.4.4	Dell Intel 32 bit chip
Msys-cl	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip
Msys-gcc			
MSVS	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip

3 The OS Library Components

3.1 OSAgent

The `OSAgent` part of the library is used to facilitate communication with remote solvers. It is not used if the solver is invoked locally (i.e. on the same machine).

3.2 OSCommonInterfaces

3.3 OSModelInterfaces

3.4 OSParsers

3.5 OSSolverInterfaces

The `OSSolverInterfaces` library is designed to facilitate linking the OS library with various solver APIs. We first describe how to take a problem instance in OSiL format and connect to a solver that has a COIN-OR OSI interface. See the OSI project `www.projects.coin-or.org/Osi`. We then describe hooking to the COIN-OR nonlinear code `Ipopt.` See `www.projects.coin-or.org/Ipopt`. Finally we describe hooking to two commercial solvers KNITRO and LINDO.

The OS library has been tested with the following solvers using the Osi Interface.

- Cbc
- Clp
- Cplex
- DyLP
- Glpk
- SYMPHONY

In the `OSSolverInterfaces` library there is an abstract class `DefaultSolver` that has the following key members:

```
std::string osil;
std::string osol;
std::string osrl;
OSInstance *osinstance;
OSResult  *osresult;
```

and the pure virtual function

```
virtual void solve() = 0 ;
```

In order to use a solver through the COIN-OR `Osi` interface it is necessary to an object in the `CoinSolver` class which inherits from the `DefaultSolver` class and implements the appropriate `solve()` function. We illustrate with the Clp solver.

```
DefaultSolver *solver  = NULL;
solver = new CoinSolver();
solver->m_sSolverName = "clp";
```

Assume that the data file containing the problem has been read into the string `osil` and the solver options are in the string `osol`. Then the `Clp` solver is invoked as follows.

```
solver->osil = osil;
solver->osol = osol;
solver->solve();
```

Finally, get the solution in `OSrL` format as follows

```
cout << solver->osrl << endl;
```

Even though LINDO and KNITRO are commercial solvers and do not have a COIN-OR `Osi` interface these solvers are used in exactly the same manner as a COIN-OR solver. For example, to invoke the LINDO solver we do the following.

```
solver = new LindoSolver();
```

Similarly for KNITRO and Ipopt. In the case of the KNITRO, the `KnitroSolver` class inherits from both `DefaultSolver` class and the KNITRO `NlpProblemDef` class. See `Kipp--putinKnitromanuallink` for more information on the KNITRO solver C++ implementation and the `NlpProblemDef` class. Similarly, for Ipopt the `IpoptSolver` class inherits from both the `DefaultSolver` class and the Ipopt `TNLP` class. See `Kipp--putinIpoptmanuallink` for more information on the Ipopt solver C++ implementation and the `TNLP` calss.

In the examples above the problem instance was assumed to be read from a file into the string `osil` and then into the class member `solver->osil.` However, everything can be done entirely in memory. For example, it is possible to use the `OSInstance` class to create an in-memory problem representation and give this representation directly to a solver class that inherits from `DefaultSolver`. The class member to use is `osinstance.` This is illustrated in the example given in Section 8.4.

3.6 OSUtils

4 OSInstance: A General Instance API

4.1 Get Methods

4.2 Set Methods

4.3 Calculate Methods

5 Hooking to An Algorithmic Differentiation Package

6 The OSSolverService

The `OSSolverService` is a command line executable designed to pass problem instances to solvers and get the result back to be displayed either to standard output or a specified browser. The `OSSovlerService` can be used to invoke a solver locally or on a remote server. It can work either synchronously or asynchronously.

6.1 OSSolverService Input Parameters

At present, the `OSSolverService` takes the following parameters. The order of the parameters is irrelevant.

-osil

The input parameters to the `OSSolverService` may be given entirely in the command line or in a configuration file. We first illustrate a configuration file input. Assume that we are interested in invoking a local solver, `Clp` to solve a linear program instance in the file `parincLinear.osil`. In the OS distribution the `OSSolverService` is the `src` directory. That is, with respect to the distribution root, it is in the directory ./OS/src/. A copy of `parincLinear.osil` is in ./OS/data. The command is:

```
OSSolverService -config ../data/testlocal.config
```

Kipp – point out that what is at the command line will override the configure file.

6.2 Solving Problems Locally

6.3 Solving Problems Remotely with Web Services

Illustrate when data is uploaded locally or when data is uploaded from another server.

7 Setting up a Solver Service with Tomcat

8 Examples

8.1 AMPL Client: Hooking AMPL to Solvers

The `amplClient` executable is designed to work with the AMPL program. See `www.ampl.com`. The `amplClient` acts like an AMPL "solver." The `amplClient` is linked with the OS library and can be used to solve problems either remotely. In both cases the `amplClient` uses the `OSnl2osil` class to convert the AMPL generated nl file (which represents the problem instance) into the corresponding instance representation in the OSiL format.

For example, assume that there is a problem instance, `hs71.mod` in AMPL model format. To solve this problem locally by calling the `amplClient` from AMPL first start AMPL and then execute the following commands. In this case we are assuming that the local solver used is `Ipopt.`

```
# take in problem 71 in Hock and Schittkowski
# assume the problem is in the AMPL directory
model hs71.mod;
# tell AMPL that the solve is amplClient
option solver amplClient;
# now tell amplClient to use Ipopt
```

```
option amplClient_options "solver ipopt";
# the name of the nl file (this is optional)
write gtestfile;
# now solve the problem
solve;
```

This will invoke Ipopt locally and the result in OSrL format will be displayed on the screen. In order to call a remote solver service, after the command

```
option amplClient_options "solver ipopt";
```

provide an option which has the address of the remote solver service.

```
option ipopt_options "http://128.135.130.17:8080/ipopt/IPOPTSolverService.jws";
```

8.2 CppAD: Using the CppAD Algorithmic Differentiation Package

8.3 File Upload: Using a File Upload Package

8.4 Instance Generator: Using the OSInstance API to Generate Instances