Optimization Services 2.3 User's Manual

Horand Gassmann, Jun Ma, Kipp Martin, and Wayne Sheng

May 6, 2011

Abstract

This is the User's Manual for the Optimization Services (OS) project. The objective of OS is to provide a general framework consisting of 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 C++ and Java source code for libraries and executable programs that implement OS standards. The OS library includes a robust solver and modeling language interface (API) for linear, nonlinear and other types of optimization problems. Also included is the C++ source code for a command line executable `OSSolverService` for reading problem instances (OSiL format, nl format, MPS format) and calling a solver either locally or on a remote server. Finally, both Java source code and a Java `war` file are provided for users who wish to set up a solver service on a server running Apache Tomcat. See the Optimization Services home page `http://www.optimizationservices.org` and the COIN-OR Trac page `http://projects.coin-or.org/OS` for more information.

Contents

List of Figures

List of Tables

1 The Optimization Services (OS) Project

The objective of Optimization Services (OS) is to provide a general framework consisting of 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 COIN-OR Trac page `http://projects.coin-or.org/OS` or the Optimization Services Home Page `http://www.optimizationservices.org` for more information.

Like other COIN-OR projects, OS has a versioning system that ensures end users some degree of stability and a stable upgrade path as project development continues. The current stable version of OS is 2.3, and the current stable release is 2.3.0, based on trunk version 3903.

The OS project provides the following:

1. A set of XML based standards for representing optimization instances (OSiL), optimization results (OSrL), and optimization solver options (OSoL). There are other standards, but these are the main ones. The schemas for these standards are described in Section 6.

2. Open source libraries that support and implement many of the standards.

3. A robust solver and modeling language interface (API) for linear and nonlinear optimization problems. Corresponding to the OSiL problem instance representation there is an in-memory object, `OSInstance`, along with a collection of `get()`, `set()`, and `calculate()` methods for accessing and creating problem instances. This is a very general API for linear, integer, and nonlinear programs. Extensions for other major types of optimization problems are also in the works. Any modeling language that can produce OSiL can easily communicate with any solver that uses the OSInstance API. The `OSInstance` object is described in more detail in Section 11. The nonlinear part of the API is based on the COIN-OR project CppAD by Brad Bell (`http://projects.coin-or.org/CppAD`) but is written in a very general manner and could be used with other algorithmic differentiation packages. More detail on algorithmic differentiation is provided in Section 13.

4. A command line executable `OSSolverService` for reading problem instances (OSiL format, AMPL nl format, MPS format) and calling a solver either locally or on a remote server. This is described in Section 7.

5. Utilities that convert AMPL nl files and MPS files into the OSiL XML format. This is described in Section 10.3.

6. Standards that facilitate the communication between clients and optimization solvers using Web Services. In Section 10.1 we describe the `OSAgent` part of the OS library that is used to create Web Services SOAP packages with OSiL instances and contact a server for solution.

7. An executable program `OSAmplClient` that is designed to work with the AMPL modeling language. The `OSAmplClient` appears as a "solver" to AMPL and, based on options given in AMPL, contacts solvers either remotely or locally to solve instances created in AMPL. This is described in Section 9.1.

8. Server software that works with Apache Tomcat and Apache Axis. This software uses Web Services technology and acts as middleware between the client that creates the instance and the solver on the server that optimizes the instance and returns the result. This is illustrated in Section 8.

9. A lightweight version of the project, `OSCommon`, for modeling language and solver developers who want to use OS API, readers and writers, without the overhead of other COIN-OR projects or any third-party software. For information on how to download `OSCommon` see Section 3.5.

2 Quick Roadmap

If you want to:

- Download the OS source code or binaries – see Section 3.
- Download just the OS API, readers and writers – see Section 3.5.
- Build the OS project from the source code – see Section 4.
- Use the OS library to build model instances or use solver APIs – see Sections 10.3, 10.5 and 11.
- Use the OSSolverService to read files in nl, OSiL, or MPS format and call a solver locally or remotely – see Section 7.
- Use AMPL to solve problems either locally or remotely with a COIN-OR solver, Cplex, GLPK, or LINDO – see Section 9.1.
- Use GAMS to solve problems either locally or remotely – see Section 9.2.
- Build a remote solver service using Apache Tomcat – see Section 8.
- Use MATLAB to generate problem instances in OSiL format and call a solver either remotely or locally – see Section 9.3.
- Use the OS library for algorithmic differentiation (in conjunction with COIN-OR CppAD) – see Section 13.
- Use modeling languages to generate model instances in OSiL format – see Section 9.

3 Downloading the OS Project

The OS project is an open-source project with source code under the Common Public License (CPL). See `http://www.ibm.com/developerworks/library/os-cpl.html`. This project was initially created by Robert Fourer, Jun Ma, and Kipp Martin. The code has been written primarily by Horand Gassmann, Jun Ma, and Kipp Martin. Horand Gassmann, Jun Ma, and Kipp Martin are the COIN-OR project leaders and active developers for the OS project. Most users will only be interested in obtaining the binaries, which we describe in Section 3.1. The remaining sections of this chapter deal with obtaining the source code for the project, which will be of interest mostly to developers.

3.1 Obtaining the Binaries

If the user does not wish to compile source code, the OS library, OSSolverService executable and Tomcat server software configuration are available in binary format for some operating systems. The repository is at `http://www.coin-or.org/download/binary/OS/`. Unlike the source code described in Section 3.3, the binary files are not subject to version control and can be downloaded using an ordinary browser. If binaries are not provided for a particular operating system, it may be possible to build them from the source code. Since the source is under version control, this requires svn. (See Sections 3.2.1, 3.3 and 4.

The binary distribution for the OS library and executables follows the following naming convention:

```
OS-version_number-platform-compiler-build_options.tgz (zip)
```

For example, OS Release 2.1.0 compiled with the Intel 9.1 compiler on an Intel 32-bit Linux system is:

```
OS-2.1.0-linux-x86-icc9.1.tgz
```

For more detail on the naming convention and examples see:

```
https://projects.coin-or.org/CoinBinary/wiki/ArchiveNamingConventions
```

After unpacking the `tgz` or `zip` archives, the following folders are available.

bin – this directory has the executables `OSSolverService` and `OSAmplClient`.

include – the header files that are necessary necessary in order to link against the OS library.

lib – the libraries that are necessary for creating applications that use the OS library.

share – license and author information for all the projects used by the OS project.

Files are also provided for an Apache Tomcat Web server along with the associated Web service that can read SOAP envelopes with model instances in OSiL format and/or options in OSoL format, call the `OSSolverService`, and return the optimization result in OSrL format. The naming convention for the server binary is

```
OS-server-version_number.tgz (.zip)
```

For example, the files associated with OS server release 2.0.0 are in the binary distribution

```
OS-server-2.0.0.tgz
```

There is no platform information given since the server and related binaries were written in Java. The details and use of this distribution are described in Section 8.

Finally for Windows users we provide Visual Studio project files (and supporting libraries and header files) for building projects based on the OS library and libraries used by the OS project. The binary for this is named

```
OS-version_number-VisualStudio.zip
```

For example, the necessary files associated with OS stable 2.3 are in the binary distribution

```
OS-2.1-VisualStudio.zip
```

The binaries provided are based on Visual Studio Express 2008. See Section 4.2.2 for more detail.

3.2 Auxiliary Software for Working with the OS Project

Compiling and modifying the OS project source code can be a daunting task, made somewhat easier by the inclusion of configure scripts and makefiles in the distribution of the source. However, additional software packages are sometimes needed or convenient, especially on Windows. We collect in this section a number of recommended packages that we ourselves use in the development and maintenance of the code.

3.2.1 Subversion (SVN)

The Subversion version control package is used to obtain the C++ source code. Users with Unix operating systems will most likely have a command line svn client. If an svn client is not present, see `http://subversion.tigris.org` to download an svn client. For Windows users we recommend the svn client TortoiseSVN. (See `http://tortoisesvn.tigris.org`.) Upon installation the TortoiseSVN client is integrated within the Windows Explorer.

3.2.2 wget

Certain third-party software (see Section 4.5) is available in source form but is not contained in the OS project distribution. Scripts are included to download this code using the `wget` executable.

A Windows version of `wget` is available at

`http://www.christopherlewis.com/WGet/wget-1.11.4b.zip`

There is no need to rebuild the code locally, which relies on several levels of other software.

3.2.3 Windows development platform

A development platform is essential for users on Windows. OS Project provides support for Microsoft Visual Studio (see Section 4.2.1) and several unix emulators, including Cygwin (Section 4.2.3), MinGW (Section 4.2.4) and MSYS (Section 4.2.5). Download instructions for all of these packages are included in the sections indicated.

3.2.4 C++ compiler

A C++ compiler is needed to compile the OS source. This should be present under all unix installations. If no C++ compiler is available on the system, the free `gcc` compiler can be downloaded from `http://gcc.gnu.org`.

Microsoft Visual Studio can be configured with the Microsoft `cl` compiler, which also works under MSYS. MinGW and Cygwin are normally configured with the Gnu compiler collection (`gcc`), although they can also be used with the `cl` compiler. However, extreme care is needed if the last option is followed. `gcc` and `cl` have very different header files, and it is important to set up the `$PATH` variable correctly in order not to confuse the header files. In our experience, best results are achieved with the minimal unix-like installation, MSYS, and the Microsoft `cl` compiler.

3.2.5 Fortran Compiler

The COIN-OR project Ipopt (see Section 4.4) and several of the third-party software described in Section 4.5 include Fortran subroutines, which must be compiled with a Fortran compiler if the user wants to include these projects in the build. A free Fortran 95 compiler can be downloaded from `http://www.g95.org`. For Fortran 77 code (which includes the Blas, HSL and Lapack projects —

but **not** Mumps) it might be sufficient to download the `f2c` translator which turns Fortran 77 code into code that can subsequently be fed into a C compiler. The `f2c` translator and the `f2c` runtime library can be downloaded from `http://www.netlib.org/f2c`. Further details are available in the file `BuildTools/compile_f2c/INSTALL`, which is part of the OS distribution.

3.2.6 `flex` and `bison`

Users who want to edit the source code in the parsers described in Section 10.4 will need the additional tools `flex` and `bison`. These can be downloaded from

`http://sourceforge.net/project/showfiles.php?group_id=2435&package_id=67879`

and are listed at the Web site as

```
bison-2.3-MSYS-1.0.11-1
flex-2.5.33-MSYS-1.0.11-1
regex-0.12-MSYS-1.0.11-1
```

The last file contains an important DLL, msys-regex-0.dll, without which `flex` will not start.

3.2.7 doxygen

Doxygen (`http://www.doxygen.org`) is a document production system that can be used to prepare documentation for the OS project and related software. For details, see Section 4.7.

3.3 Obtaining OS Source Code Using Subversion (SVN)

For the rest of this documentation, we assume that the name of the root directory of the OS project distribution is `COIN-OS`. The `COIN-OS` directory structure is illustrated in Figure 1. OS source code is mainly contained inside of the OS subdirectory. Other first level subdirectories are mostly external projects (COIN-OR or third-party) that the OS project depends on.

For Users on a Unix system such as Linux, Solaris, Mac OS X, etc., the source code is obtained as follows. In a command window execute:

```
svn co https://projects.coin-or.org/svn/OS/releases/2.3.0 COIN-OS
```

It is possible that on some systems you may get a message such as:

```
Error validating server certificate for 'https://projects.coin-or.org:443':
 - The certificate is not issued by a trusted authority. Use the
   fingerprint to validate the certificate manually!
Certificate information:
 - Hostname: projects.coin-or.org
 - Valid: from Jun 10 22:51:18 2007 GMT until Jun 15 21:00:28 2009 GMT
 - Issuer: 07969287, http://certificates.godaddy.com/repository, GoDaddy.com, Inc.,
Scottsdale, Arizona, US
 - Fingerprint: f7:26:0f:bb:e1:94:a5:23:7f:5c:cb:c3:9a:c4:74:51:e5:c7:4d:29
(R)eject, accept (t)emporarily or accept (p)ermanently?
```

If so, select `(p)` and you should not get this message again.

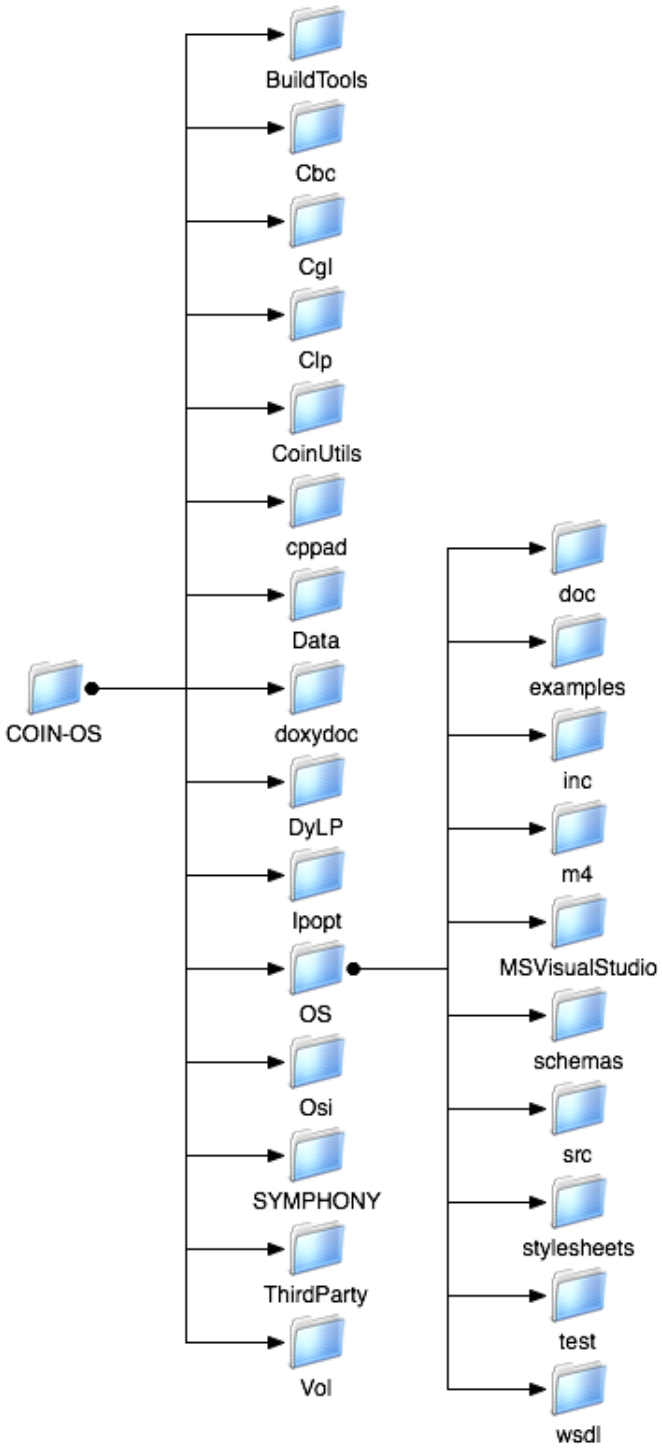

Figure 1: The OS distribution root directory.

For more information on downloading the OS project or other COIN-OR projects using SVN see

`http://projects.coin-or.org/BuildTools/wiki/user-download#DownloadingtheSourceCode`.

On Windows with TortoiseSVN, create a directory `COIN-OS` in the desired location and right-click on this directory. Select the menu item `SVN Checkout ...` and in the textbox "`URL of Repository`" give the URL for the version of the OS project you wish to check out, for instance,

`https://projects.coin-or.org/svn/OS/stable/2.3`.

Now build the project as described in Section 4.

The Java source code for setting up a solver service with Apache Tomcat is checked out as follows:

```
svn co https://projects.coin-or.org/svn/OS/branches/OSjava OSJava
```

For more detail on running a Tomcat solver service see Section 8.

3.4 Obtaining the OS Source Code From a Tarball or Zip File

The OS source code can also be obtained from either a tarball or zip file. This may be preferred for users who are not managing other COIN-OR projects and wish to only work with periodic release versions of the code. In order to obtain the code from a Tarball or Zip file do the following.

Step 1: In a browser open the link `http://www.coin-or.org/download/source/OS/`. Listed at this page are files in the format:

```
OS-release_number.tgz
OS-release_number.zip
```

Step 2: Click on either the `tgz` or `zip` file and download to the desired directory.

Step 3: Unpack the files. For `tgz` do the following at the command line:

```
gunzip OS-release_number.tgz
tar -xvf OS-release_number.tar
```

Windows users should be able to double-click on the file `OS-release_number.zip` and have the directory unpacked.

Step 4: (optional) Move the folder `OS-release_number` to the desired location and rename it to `COIN-OS`.

Now build the project as described in Section 4.

3.5 Obtaining source for the OS Project API

The OS project is very extensive and relies on many other COIN-OR projects. This may not be desirable for modeling language and solver developers who just wish to use the OS API in conjunction with their modeling language or solver. Hence there is also an "OS lite" download that consists of all the code for the OS API and for reading and writing instance and solution files. We refer to this version of the project as `OSCommon`. To get the current version of `OSCommon` use the svn command

```
svn co https://projects.coin-or.org/svn/OS/branches/OScpp/OSCommon OSCommon
```

4 Building and Testing the OS Project

Once the OS source code is obtained, the OS libraries, `OSSolverService` executable, and test examples can be built. We describe how to do this on Unix/Linux systems (see Section 4.1) and on Windows (see Section 4.2).

4.1 Building the OS Project on Unix/Linux Systems

In order to build the OS project on Unix/Linux systems do the following.

Step 1: Connect to the OS distribution root directory (`COIN-OS` in Figure 1).

Step 2: Run the configure script that will generate the makefiles. If you are running on a machine with a Fortran 95 compiler present (e.g., `gfortran`), and you have previously downloaded the third-party software packages `BLAS` and `Mumps` (see Section 4.4), run the command

```
./configure
```

otherwise use

```
./configure  COIN_SKIP_PROJECTS="Ipopt Bonmin"
```

as COIN-OR's `Ipopt` and `Bonmin` projects currently use Fortran to compile some of its dependent libraries.

Notes:

- If `gfortran` is not present and you wish to build the nonlinear solver `Ipopt` see the instructions in Section 4.4.
- When using `configure` you may wish to use the `-C` option. This instructs `configure` to use a cache file, `config.cache`, to speed up configuration by remembering and reusing the results of tests already performed.
- For more information and options on the `./configure` script see `https://projects.coin-or.org/BuildTools/wiki/user-configure#PreparingtheCompilation`.
- You cannot apply `COIN_SKIP_PROJECTS` to `Cbc`, `Clp`, `Cgl`, `CoinUtils`, or `Osi`. These projects must be present.

Step 3: Run the make files.

```
make
```

Step 4: Run the `unitTest`.

```
make test
```

Depending upon which third-party software you have installed, the result of running the `unitTest` should look something like (we have included the third-party solver LINDO in the test results below; it is not part of the default build):

```
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 p0033.osil with SYMPHONY
Solved problem parincLinear.osil with DyLP
Solved problem volumeTest.osil with Vol
Solved problem p0033.osil with GLPK
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 using 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

All tests completed successfully
```

If you do not see

```
All tests completed successfully
```

then you have not passed the unitTest and hopefully some semi-intelligible error message was given.

Step 5: Install the libraries and executables.

```
make install
```

This will install all of the libraries in the `lib` directory. In particular, the main OS library `libOS` along with the libraries of the other COIN-OR projects that download with the OS project will get installed in the `lib` directory. In addition the `make install` command will install several executable programs in the `bin` directory, depending on the parameters on the `configure command`. One of these binaries is `OSSolverService` which is the main OS project executable. This is described in Section 7. In addition `Clp`, `Cbc`, `Ipopt`, `Bonmin`, `Couenne` and `SYMPHONY` get installed in the `bin` directory. Necessary header files are installed in the `include` directory. In this case, `bin`, `lib` and `include` are all subdirectories of where `./configure` is run. If the user wants these files installed elsewhere, then `configure` should specify the `prefix` of these directories. That is,

```
./configure  --prefix=prefixDirectory  COIN_SKIP_PROJECTS="Ipopt Bonmin"
```

For example, running

```
./configure  --prefix=/usr/local  COIN_SKIP_PROJECTS="Ipopt Bonmin"
```

and then running `make` and `make install` will put the relevant files in

```
/usr/local/bin
/usr/local/include
/usr/local/lib
```

Run an Example! If `make test` works, proceed to Section 7 to run the key executable, `OSSolverService`.

4.1.1 Building the OS Project on Mac OS X

When building OS on Mac OS X 10.5.x (Leopard) it may be necessary to add the following to the configure line

```
ADD_CXXFLAGS="-mmacosx-version-min=10.4"
ADD_CFLAGS="-mmacosx-version-min=10.4"
ADD_FFLAGS="-mmacosx-version-min=10.4"
LDFLAGS="-flat_namespace"
```

Also, the Mac OS X operating system does not come configured with the gcc compiler. Users wanting to build the OS project on the Mac should do the following:

- Install the Xcode developer tools. These are available on the install DVD that comes with the machine or at the Apple developer site. See

  `http://developer.apple.com/technology/xcode.html`

- Install a Fortran compiler. We have had good luck with the GNU **gfortran** compiler. Platform specific binaries for the various Mac platforms (Leopard and Tiger, Intel and Power PC) are obtained at

  `http://hpc.sourceforge.net/`

  We followed the instructions and installed the binary using the command

  ```
  sudo tar -xvf gcc-bin.tar -C /
  ```

We have also successfully used the fink project, see
`http://www.finkproject.org/`
to download and build gcc/g++/gfortran compilers from source code.

4.2 Building the OS Project on Windows

There are a number of options open to Windows users. First, if you wish to work with source code we recommend downloading the svn client, TortoiseSVN. (See Section 3.2.1.) With TortoiseSVN in the Windows Explorer connect to the directory (e.g., COIN-OS) where you wish to put the OS code. Right-click on the directory and select `SVN Checkout`. In the textbox, `URL of Repository` give the URL for the version of the OS project you wish to check out, e.g.,

`https://projects.coin-or.org/svn/OS/stable/2.3`.

Also, if you plan to build any of the projects contained in `ThirdParty` (e.g., ASL) we recommend using `wget`. (See Section 3.2.2.)

4.2.1 Microsoft Visual Studio (MSVS)

Microsoft Visual Studio solution and project files are provided for users of Windows and the Microsoft Visual Studio IDE. We currently support Versions 8 and 9. These versions are also sometimes referred to by their (approximate) release dates, which is 2008 for Version 9 and 2005 for Version 8. In addition there is a free version of the Visual Studio IDE C++ compiler, called Visual C++ Express Edition.

The following steps are necessary to build the OS project using the Microsoft Visual Studio IDE.

Step 0. If the C++ compiler `cl` is already installed, go to to Step 2.

Step 1. Download and install the Visual C++ Express Edition, which is available for free at Microsoft's web site. Version 9 is at `http://www.microsoft.com/express/download/#webInstall`. This download contains the Microsoft `cl` C++ compiler along with necessary libraries.

Step 2. The part of the OS library responsible for communication with a remote server depends on some underlying Windows socket header files and libraries. These files are part of the commercial for-pay version, but are not included in the Visual C++ Express download. If you have the Express Edition, it is necessary to also download and install the Windows Platform SDK, which can be found at

`http://www.microsoft.com/downloads/details.aspx?FamilyID=E6E1C3DF-A74F-4207-8586-711EBE331CDC&displaylang=en.`

Step 3. In the COIN-OR/OS directory you will find the folder MSVisualStudio, which contains root directories organized by the version of Visual Studio. We currently provide solution files for Version 8 and Version 9. Each contains the file `OS.sln` and project files for building the unitTest (`OSTest.vcproj`), the OSSolverService (`OSSolverService.vcproj`) and the OS libraries (`libOSCommon.vcproj` and (`libOSSolvers.vcproj`). The Microsoft Visual Studio files are automatically downloaded with an SVN checkout. They are also contained in the tarballs (see Section 3.4).

Open the solution file or the individual project files (for instance by double-clicking on them in Windows Explorer) and select Build from the menu bar.

Step 4. Run the `unitTest`. Connect to the directory `COIN-OR/OS/test` and run either the release or debug version of the `unitTest` executable.

The solution file `OS.sln` contains two configurations, `Debug` and `Release`, both of which are configured without `Ipopt`.

4.2.2 Visual Studio Examples Distribution

Many users will not be interested in actually building the OS project from source code. At the link `https://projects.coin-or.org/CoinBinary/browser/binary/OS` are binaries for using the OS project. There are also Visual Studio project files for building applications that use the precompiled OS libraries. In particular, download and unpack the file

`OS-version_number-VisualStudio.zip`

This zip archive contains a `bin` directory that holds the executable `OSSolverService.exe`. The `OSSolverService.exe` is configured to run, out-of-the-box, the following solvers.

- Bonmin
- Clp
- Cbc
- Couenne
- DyLP
- Ipopt
- SYMPHONY
- Vol

The libraries necessary to run these solvers are included in the download. *No additional software is necessary to solve models with these solvers!* See Section 7 for details on how to use the `OSSolverService.exe` executable for solving optimization problems.

The `bin` directory also contains the `OSAmplClient.exe` executable. If the user has a Windows version of AMPL, then AMPL can be used to invoke all of the solvers mentioned above through the `OSAmplClient`. For details see Section 9.1.

This zip archive also contains a `lib` directory that holds libraries for a number of COIN-OR projects, including OS. It is possible to build customized optimization applications that link against

these libraries. We provide several examples that use various aspects of the OS project in order to build customized applications. The Visual Studio example solution file is named `osExamples.sln` and is found in the folder `MSVisualStudioOSExamples`. The solution file `osExamples.sln` currently contains nine projects (examples). These are described in more detail in Section 12.

4.2.3 Cygwin

`Cygwin` provides a Unix emulation environment for Windows. It comes with numerous tools and libraries including the `gcc` compilers. See `www.cygwin.com`. Cygwin can be used with the Gnu Compiler Collection (`gcc`) or with the Microsoft `cl` compiler.

Using Cygwin with `gcc`: With Cygwin and the corresponding `gcc` compiler the OS project is built exactly as described in Section 4.1. If you previously downloaded Cygwin with gnome make version 3.81-1, you must obtain a fixed 3.81 version from `http://www.cmake.org/files/cygwin/make.exe`. (See also the discussion at `http://projects.coin-or.org/BuildTools/wiki/current-issues`.)

Using Cygwin with Microsoft `cl`: Users who are extremely adventuresome and have an abundance of free time on their hands may wish to use Cygwin with the Microsoft `cl` compiler to build the OS project. The following steps have led to a successful build.

Step 1: Download `Cygwin` from `http://www.cygwin.com/setup.exe` and install.

Step 2: Download Visual Studio Express C++ at

`http://www.microsoft.com/express/download/#webInstall`.

Step 3: The part of the OS library responsible for communication with a remote server depends on some underlying Windows socket header files and libraries. Therefore it is necessary to also download and install the Windows Platform SDK. Download the necessary files at

`http://www.microsoft.com/downloads/details.aspx?FamilyID=E6E1C3DF-A74F-4207-8586-711EBE331CDC&displaylang=en`

and install.

Step 4: Set the Cygwin search path configuration. This is important. This step is necessary to ensure that Cygwin looks for compilers, linkers, etc in the correct order. The right order of directories is: MSVS command directories, Cygwin command directories, and finally Windows command directories. This is illustrated below.

- First, Cygwin should look in the Microsoft Visual Studio directories. If a standard Visual Studio install is done, the following should be part of the Cygwin search path.

```
.
:/cygdrive/c/Program Files/Microsoft Visual Studio 8/Common7/IDE
:/cygdrive/c/Program Files/Microsoft Visual Studio 8/VC/bin
:/cygdrive/c/Program Files/Microsoft Visual Studio 8/Common7/Tools
:/cygdrive/c/Program Files/Microsoft Visual Studio 8/SDK/v2.0/Bin
:/cygdrive/c/Program Files/Microsoft Visual Studio 8/VC/vcpackages
:/cygdrive/c/WINDOWS/Microsoft.NET/Framework/v2.0.50727
```

- Second, Cygwin should next search its command directories. The following is typical of a standard install.

```
/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin
```

- Third, Cygwin should search the Windows specific command directories. The following is typical.

```
:/cygdrive/c/WINDOWS/system32:/cygdrive/c/WINDOWS
:/cygdrive/c/WINDOWS/System32/Wbem:/cygdrive/c/Program Files/ATI Technologies/ATI Control Panel
:/cygdrive/c/Program Files/Common Files/Roxio Shared/DLLShared/
:/cygdrive/c/Program Files/QuickTime/QTSystem/:/cygdrive/c/Program Files/Microsoft SQL Server/90/Tools/bin/
:/cygdrive/c/Program Files/Microsoft Platform SDK for Windows Server 2003 R2/Bin/
:/cygdrive/c/Program Files/Microsoft Platform SDK for Windows Server 2003 R2/Bin/WinNT/
:/cygdrive/c/Program Files/SSH Communications Security/SSH Secure Shell
:/cygdrive/d/SSH
```

Open the Cygwin shell and check the value of `$PATH`. If directories don't appear in an order described above, then the `$PATH` value needs to be reset.

Step 5: Build the OS project (or any COIN-OR project). If you wish to avoid the FORTRAN related issues you should build without `Ipopt`, `Bonmin` and `Couenne`. Issue the following command in the project root.

```
./configure COIN_SKIP_PROJECTS="Ipopt Bonmin Couenne" --enable-doscompile=msvc
```

If you wish to build with `Ipopt` or `Bonmin` and `Couenne`, which depend on it, then FORTRAN is required — and Visual Studio does not ship with a FORTRAN compiler. The following is a work-around. (See also Section 4.4.)

Step a. Obtain one of the Harwell Subroutine Library (HSL) routines `ma27ad.f` or `MA57ad.f`. See `http://www.cse.scitech.ac.uk/nag/hsl/`. Put the Harwell code in the directory `ThirdParty/HSL`. (Note the case in the file names, which is relevant in a unix-like environment.)

Step b. Follow the instructions for downloading and installing the `f2c` compiler from Netlib. The installation instructions for this are in the `INSTALL` file in

`BuildTools/compile_f2c`

Step c. Run the configure script

```
./configure  --enable-doscompile=msvc
```

4.2.4 MinGW

MinGW (Minimalist GNU for Windows) is a set of runtime headers to be used with the GNU `gcc` compilers for Windows. See `www.mingw.org`. As with Cygwin, the OS project is built exactly as described in Section 4.1.

The MinGW installation includes the `gcc` compiler, which can interact negatively with the Microsoft `cl` compiler. For that reason it is advisable to download the even smaller installation MSYS (see next section) if you intend to build any software with the Microsoft Visual Studio suite.

Warning: A user of MSYS with MinGW gcc version 4.4.0 got an error about a missing library "pthreadsGC2.dll" when running the OS `unitTest`. This user installed `pthreadsGC2.dll` from

`ftp://sources.redhat.com/pub/pthreads-win32/dll-latest/lib/pthreadGC2.dll`

and reported that the problem then went away.

4.2.5 MSYS

MSYS (Minimal SYStem) provides an easy way to use the COIN-OS build system with compilers/linkers of your own choice, such as the Microsoft command line C++ `cl` compiler. MSYS is intended as an alternative to the DOS command window. It is an application that gives the user a Bourne shell that can run `configure` scripts and makefiles. No compilers come with MSYS. In the Cygwin, MinGW, and MSYS hierarchy, it is at the bottom of the food chain in terms of tools provided. However, it is very easy to use and build the OS project with MSYS. In this discussion we assume that the user has downloaded the OS source code (most likely with TortoiseSVN) and that the `cl` compiler is present. The project is built using the following steps.

Note:

- If you wish to use the third-party software with MSYS it is best to get `wget`. See Section 3.2.2.
- Do not put any imbedded blanks in the path to the OS project.

Execute the following steps to use the Microsoft C++ `cl` compiler with MSYS.

Step 1. Download `MSYS` at

`http://downloads.sourceforge.net/mingw/MSYS-1.0.11.exe?modtime=1079444447&big_mirror=1`

and install. Double-clicking on the MSYS icon will open a Bourne shell window.

Step 2. Download Visual Studio Express C++ at

`http://www.microsoft.com/express/download/#webInstall`

and install.

Step 3. The part of the OS library responsible for communication with a remote server depends on some underlying Windows socket header files and libraries. Therefore it is necessary to also download and install the Windows Platform SDK. Download the necessary files at

`http://www.microsoft.com/downloads/details.aspx?FamilyID=E6E1C3DF-A74F-4207-8586-711EBE331CDC&displaylang=en`

and install.

Step 4. Set the Visual Studio environment variables so that paths to the necessary libraries and header files are recognized. Assuming that a standard installation was done for the Visual Studio Express and the Windows Platform SDK set the variables as follows:

```
PATH=C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;
C:\Program Files\Microsoft Visual Studio 8\VC\BIN;
C:\Program Files\Microsoft Visual Studio 8\Common7\Tools;
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\bin;
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;
C:\Program Files\Microsoft Visual Studio 8\VC\VCPackages


INCLUDE=C:\Program Files\Microsoft Visual Studio 8\VC\INCLUDE;
C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Include

LIB = C:\Program Files\Microsoft Visual Studio 8\VC\LIB;
```

```
C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\lib;
C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\Lib
```

The environment variables can be set using the `System Properties` in the Windows `Control Panel`.

Step 5. In the MSYS command window connect to the root of the OS project and run the `configure` script followed by `make` as described in Section 4.1.

Run an Example! If `make test` works, proceed to Section 7 to run the key executable, `OSSolverService`.

Microsoft Windows users who wish to obtain MSYS for building the OS project can download the appropriate software at `http://sourceforge.net/project/showfiles.php?group_id=2435`. The user may find this Web site confusing. It is only necessary to download what is referred to as the **MSYS Base System**. As of this writing the most recent version is MSYS-1.0.11. This file is listed as `bash-3.1-MSYS-1.0.11` and the binary download is

`http://downloads.sourceforge.net/mingw/bash-3.1-MSYS-1.0.11-1.tar.bz2?modtime=1195140582&big_mirror=1`

This will provide the necessary Bourne shell for executing the configure scripts. Users who want to edit the source code in the parsers described in Section 10.4 will need the additional tools **flex** and **bison** as described in Section 3.2.6.

4.3 VPATH Installations

It is possible to build the OS project in a directory that is different from the directory where the source code is present. This is called a `VPATH` compilation. A `VPATH` compilation is very useful if you wish to build several versions (e.g., debug and non-debug versions, or versions with availability of various combinations of third-party software) of the OS project from a single copy of the source code.

For example, assume you wish to build a debug version of the OS project in the directory `vpath-debug` and that `../COIN-OS` is the path to the root of the OS project distribution. Create the `vpath-debug` directory, leaving it empty for the moment. From the `vpath-debug` directory, run `configure` as follows:

```
../COIN-OS/configure --enable-debug
```

After you run `configure`, the OS distribution directory structure (see Figure 1) will be mirrored in the `vpath-debug` directory, and all of the necessary `Makefile`s will be copied there. Next from the `vpath-debug` directory execute

```
make
```

and all of the libraries created will be in their respective directories inside `vpath-debug` and not `../COIN-OS`.

Notes:

1. If you have already run the `configure` script inside the `../COIN-OS` directory, you cannot do a `VPATH` build until you have run

   ```
   make distclean
   ```

in the `../COIN-OS` directory.

2. Note also that `configure` automatically detects the presence of third-party software and prepares the configuration and make files accordingly. Once you have downloaded, e.g., Blas, you must specify

   ```
   configure COIN_SKIP_PROJECTS="ThirdParty/Blas"
   ```

   if you want to recreate the default configuration.

3. If you work with the trunk version of OS, it is possible that files are added to and removed from the distribution due to development activities. These files are not recognized properly by the system unless it is reconfigured by running

   ```
   make distclean
   ```

   followed by

   ```
   ./configure
   ```

   in the `VPATH` directory.

4. You can customize compiler flags, linker options, include directories, and many other options by setting appropriate environment variables. For further information you may want to consult the built-in help function by specifying

   ```
   ./configure --help
   ```

   or the help file available at the homepage of the `BuildTools` project (http://projects.coin-or.org/BuildTools).

4.4 COIN-OR Projects Requiring Fortran

Ipopt, Bonmin and Couenne are COIN-OR projects (`http://projects.coin-or.org/Ipopt`, `http://projects.coin-or.org/Bonmin`, `http://projects.coin-or.org/Couenne`) and are included in the download with the OS project. However, unlike the other COIN-OR projects that download with OS, these projects require third-party software that is based on FORTRAN and is *not* part of the default distribution. Care must therefore be taken if you wish to build OS with the Ipopt, Bonmin or Couenne solver. It is further important to know that there is a dependency between these three projects. Ipopt is the only one using Fortran directly, but Bonmin relies on Ipopt for its solver, and Couenne is similarly dependent on both Ipopt and Bonmin. Neither Bonmin nor Couenne can therefore be installed in isolation.

You can exclude all three of these projects from the OS build by adding the option

```
COIN_SKIP_PROJECTS="Ipopt Bonmin Couenne"
```

to the `configure` script.

4.4.1 Building Ipopt, Bonmin and Couenne in Unix or a Unix-like environment

If you are working in Unix or one of the Unix-like environments described in section 4.2, you can proceed as follows. To get the necessary third-party software, first connect into the `ThirdParty` directory. Then execute the following commands:

```
$ cd Blas
$ ./get.Blas
$ cd ../Lapack
$ ./get.Lapack
$ cd ../Mumps
$ ./get.Mumps
```

What you do next depends upon whether or not a FORTRAN compiler is present, and if so, which version of FORTRAN. There are several options. See also

`http://www.coin-or.org/Ipopt/documentation/node13.html`

Option 1. If you have a Fortran 95 compiler that recognizes embedded preprocessor statements (such as `gfortran` — see `http://gcc.gnu.org/fortran/` or `g95` — see `http://www.g95.org`), you can simply run the `configure` script and the FORTRAN compiler will be detected and the `Ipopt`, `Bonmin` and `Couenne` projects will be built.

Option 2. If your Fortran 95 compiler cannot deal with the preprocessor statements embedded in the Mumps code, it may be possible to run the Fortran code through a preprocessor such as `cpp`. In the worst case you may have to resort to manual edits before you can build Ipopt — or see Option 3.

Option 3. If you have a FORTRAN 77 compiler, you can replace Mumps by one of the Harwell Subroutine Library (HSL) routines `ma27ad.f` or `MA57ad.f`. (Unix is case-sensitive, so note the file names carefully.) See

`http://www.cse.scitech.ac.uk/nag/hsl/`.

You must obtain the Harwell code and put it in the directory `./ThirdParty/HSL`. Now run the `configure` script as described in Section 4.1.

Note that the Harwell Subroutine Library is not governed by the Common Public License. It is the user's responsibility to ensure adherence to appropriate copyright and distribution agreements.

Option 4. If you do not have a FORTRAN compiler and do not wish to obtain one, you can use the `f2c` translator from Netlib to translate HSL to `C`. The installation instructions for `f2c` are in the `INSTALL` file in

`BuildTools/compile_f2c`

Two important points:

- Option 4 also requires that one of the Harwell Subroutine Library (HSL) routines `ma27ad.f` or `MA57ad.f` be present in the HSL directory.
- If you run `configure` with the `--enable-debug` option on Windows, then when building the `vcf2c.lib`, use the command line

```
CFLAGS = -MTd -DUSE_CLOCK -DMSDOS -DNO_ONEXIT
```

4.4.2 Ipopt and Microsoft Visual Studio

We regret that at present we cannot distribute a solution file that can detect and reliably process the necessary third-party software to build Ipopt. Users who need Ipopt on a Windows system are advised to download the binary build as documented in Section 3.1.

4.5 Other Third-Party Software

This section deals with other third-party software not available for download at `www.coin-or.org`. The OS project distribution includes the COIN-OR projects `Bonmin`, `Cbc`, `Clp`, `Cgl`, `CoinUtils`, `Couenne`, `CppAD`, `DyLP`, `Ipopt`, `Osi`, `SYMPHONY`, and `Vol`. (For details on any of these projects see the COIN-OR web site at `http://www.coin-or.org/projects/`.) However, the project is also designed to work with several other open source and commercial software projects. In the OS distribution directory structure (see Figure 1), there is a `ThirdParty` directory, which does not contain anything other than `get.xxxx` scripts and other utilities. The source code for any of these packages must be downloaded separately using the `get.xxxx` scripts, as `configure` will not build these projects without the source code being present. After the download, `configure` will recognize the presence of these files and will configure the makefiles accordingly.

If the user wants to exclude these projects from the build after they have been downloaded and detected, a new `configure` is required with instructions to skip them. For instance, if the user experiences problems with the Fortran compiler and its interaction with the system, the following command can be used to skip all projects that use Fortran code:

```
configure COIN_SKIP_PROJECTS="Ipopt Bonmin Couenne ThirdParty/Blas ThirdParty/Lapack \
ThirdParty/Mumps"
```

In the `inc` subdirectory of the `OS` directory, there is a header file, `config_os.h` that defines the values of a number of

```
COIN_HAS_XXXXX
```

variables.

Many of the other header files contain `#include` statements inside `#ifdef` statements. For example,

```
#ifdef COIN_HAS_LINDO
#include "LindoSolver.h"
#endif
#ifdef COIN_HAS_GLPK
#include <OsiGlpkSolverInterface.hpp>
#endif
```

If the project is configured with the simple `./configure` command given in Step 2 on page 13 with no arguments, then in the `config_os.h` header file the variables associated with the third-party software described in this subsection will be undefined. For example:

```
/* Define to 1 if the Cplex package is used */
/* #undef COIN_HAS_CPX */
```

unlike the configured COIN-OR projects that appear as

```
/* Define to 1 if the Clp package is used */
#define COIN_HAS_CLP 1
```

In the following subsections we describe how to incorporate various third-party packages into the OS project and see to it that the

`COIN_HAS_XXXXX`

variable is defined in `config_os.h`.

Make sure to run `configure` after you have downloaded the required source code, in order to modify the makefiles appropriately. It is **important to note** that even though there are multiple files named `configure` in various subdirectories, you should only ever run the master configure in the distribution root directory, possibly accessed from a `VPATH` as in Section 4.3. It sets important global variables and will call all other necessary configure files in turn. You may also wish to view `http://projects.coin-or.org/BuildTools/wiki/user-configure#CommandLineArgumentsforconfigure`

for more information on command line arguments that are illustrated in the subsections below.

4.5.1 AMPL Solver Library (ASL)

The OS library contains a class, `OSnl2osil` (see Section 10.3.2), and the program `OSAmplClient` (see Section 9.1) that require the use of the AMPL Solver Library (ASL). See `http://www.ampl.com` and `http://netlib.sandia.gov/ampl/`. Users with a Unix system should locate the `ASL` folder that is part of the distribution. The `ASL` folder is in the `ThirdParty` folder which is in the distribution root folder. Locate and execute the `get.ASL` script. Do this prior to running the `configure` script. The `configure` script will then build the correct ASL library.

Microsoft Visual Studio users should note that `OSAmplClient` is distributed as part of the binary distribution. For reasons explained in Section 4.4.2 it is currently not possible to distribute a solution file to let users build their own executable.

4.5.2 GLPK

`GLPK` is a an open-source linear and integer-programming solver from the GNU organization. See `http://www.gnu.org/software/glpk/`. GLPK is distributed under the GNU General Public Licence (GPL), which is incompatible with the Common Public License (CPL) that governs OS. For that reason we are unable to distribute OS binaries linked to the GLPK solver. Users interested in GLPK must build OS from source and link to the GLPK libraries.

In order to use GLPK with OS in a unix environment, connect to `ThirdParty/Glpk` and execute `get.Glpk`. Once the source code has been downloaded, run `configure`, followed by a `make`, as explained in Section 4.1 or Section 4.3.

Users on MSVS can download the source by anonymous `ftp` from

`ftp://ftp.gnu.org/gnu/glpk/glpk-version_number.tar.gz`

At the time of this writing, the most up-to-date version is 4.42, which can be found at `ftp://ftp.gnu.org/gnu/glpk/glpk-4.42.tar.gz`

4.5.3 Cplex

Cplex is a linear, integer, and quadratic solver. See `http://www.ilog.com/products/cplex/`. Cplex does not provide source code and you can only download the platform dependent binaries. After installing the binaries and include files in an appropriate directory, run `configure` to point to the include and library directory. An example is given below:

```
configure --with-cplex-lib="-L$(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT) $(CPLEX_LIBS)"
--with-cplex-incdir= $(CPLEXDIR)/include
```

You may also need the following environment variables (if they are not already set). The following are values we used in a working implementation.

```
SYSTEM =i86_linux2_glibc2.3_gcc3.2
LIBFORMAT =static_pic_mt
CPLEXDIR =/usr/local/ilog/cplex81/include/ilcplex
CPLEXLIBPATH= -L$(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)
CPLEXINCDIR = $(CPLEXDIR)/include
CPLEX_LIBS=-lcplex -lilocplex -lm -lpthread
ILOG_HOME=/usr/local/ilog/cplex81/bin/i86_linux2_glibc2.3_gcc3.2
ILOG_LICENSE_FILE=/usr/local/ilog/ilm/access.ilm
PATH=***:/usr/local/ilog/cplex81/bin/i86_linux2_glibc2.3_gcc3.2:***
CLASSPATH=:/usr/local/ilog/cplex81/bin/i86_linux2_glibc2.3_gcc3.2:
```

4.5.4 LINDO

LINDO is a commercial linear, integer, and nonlinear solver. See `http://www.lindo.com`. LINDO does not provide source code and you can only download the platform dependent binaries. After installing the binaries and include files in an appropriate directory, run `configure` to point to the include and library directory. An example is given below:

```
configure --with-lindo-incdir=/home/kmartin/files/code/lindo/linux/include
--with-lindo-lib="-L/home/kmartin/files/code/lindo/linux/lib -llindo -lmosek"
```

4.5.5 MATLAB

MATLAB is a commercial programing environment especially suited for the development and testing of computationally intensive tasks. (See `http://www.mathworks.com/products/matlab`.) Install MATLAB on the client machine and follow the instruction in Section 9.3.

4.5.6 Library Paths

After running `configure` as described above, on Unix systems, it will be necessary to set the environment variables `LD_LIBRARY_PATH` or `DYLD_LIBRARY_PATH` (on Mac OS X) to point to the location of the installed third-party libraries in the case that the libraries are dynamic and not static libraries.

4.6 Bug Reporting

Bug reporting is done through the project Trac page. This is at

`http://projects.coin-or.org/OS`

To report a bug, you must be a registered user. For instructions on how to register, go to

`http://www.coin-or.org/usingTrac.html`

After registering, log in and then file a trouble ticket by going to

`http://projects.coin-or.org/OS/newticket`

4.7 Documentation

If you have Doxygen (`http://www.doxygen.org`) available (the executable `doxygen` should be in the `path` command) then executing

```
make doxydoc
```

in the project root directory will result in the Doxygen documentation being generated and stored in the `doxydoc` folder in the project root.

In order to view the documentation, open a browser and open the file

```
projectroot/doxydoc/html/index.html
```

By default, running Doxygen will generate documentation for only the OS project. Documentation will not be generated for the other COIN-OR projects in the project root. In the `doxydoc` folder is a configuration file `doxygen.conf`. This configuration file contains the `EXCLUDE` parameter

```
EXCLUDE =  Bonmin \
   Cbc\
   Cgl \
   Clp \
   CoinUtils \
   Couenne \
   cppad \
   SYMPHONY \
   Vol \
   DyLP \
   ThirdParty \
   Osi \
   include
```

This file can be edited, and any project for which documentation is desired, can be deleted from the `EXCLUDE` list.

4.8 Platforms

The build process described in Section 4.1 has been tested on Linux, Mac OS X, and on Windows using MinGW/MSYS and Cygwin. The `gcc/g++` and Microsoft `cl` compiler have been tested. A number of solvers have also been tested with the OS library. For a list of tested solvers and platforms see Table 1. More detail on the platforms listed in Table 1 is given in Table 2. For a list of other platforms testing the OS project see

`https://projects.coin-or.org/TestTools/wiki/NightlyBuildInAction`.

5 The OS Project Components

The directories in the project root are outlined in Figure 1.

If you download the OS package, you get these additional COIN-OR projects. The links to the project home pages are provided below and give more information on these projects.

- `Bonmin` - `http://projects.coin-or.org/Bonmin`

Table 1: Tested Platforms for Solvers

	Mac	Linux	Cyg-gcc	Msys-cl	MinGW-gcc	MSVS
Bonmin	x	x	x	x	x	x
Cbc	x	x	x	x	x	x
Cgl	x	x	x	x	x	x
Clp	x	x	x	x	x	x
Couenne	x	x		x	x	
Cplex		x				
DyLP	x	x	x	x	x	x
Glpk	x	x	x	x	x	
Ipopt	x	x	x	x	x	x
Lindo	x	x		x		x
MATLAB	x					
OSAmplClient	x	x		x		x
SYMPHONY	x	x	x	x	x	x
Vol	x	x	x	x	x	x

Table 2: Platform Description

	Operating System	**Compiler**	**Hardware**
Mac	Mac OS X 10.4.9	gcc 4.0.1	Power PC
Mac	Mac OS X 10.4.10	gcc 4.0.1	Intel
Linux	Ubuntu 7.10	gcc 4.1.2	Dell Intel 32 bit chip
Cyg-gcc	Windows 2003 Server	gcc 4.2.2	Dell Intel 32 bit chip
Msys-cl	Windows XP	cl 14.00	Dell Intel 32 bit chip
MinGW-gcc	Windows XP	gcc 3.4.2	Dell Intel 32 bit chip
MSVS	Windows XP	Visual Studio 8 and 9	Dell Intel 32 bit chip

- BuildTools - http://projects.coin-or.org/BuildTools
- Cbc - http://projects.coin-or.org/Cbc
- Cgl - http://projects.coin-or.org/Cgl
- Clp - http://projects.coin-or.org/Clp
- CoinUtils - http://projects.coin-or.org/CoinUtils
- Couenne - http://projects.coin-or.org/Couenne
- CppAD - http://projects.coin-or.org/CppAD
- DyLP - http://projects.coin-or.org/DyLP
- Ipopt - http://projects.coin-or.org/Ipopt
- Osi - http://projects.coin-or.org/Osi
- SYMPHONY - http://projects.coin-or.org/SYMPHONY
- Vol - http://projects.coin-or.org/Vol

The following directories are also in the project root.

- bin - after executing make install the bin directory will contain OSSolverService, clp, cbc and symphony.
- Data - this directory contains numerous test problems that are used by the unit tests of the COIN-OR projects just mentioned.
- doxydoc - is a folder for documentation.
- include - is a directory for header files. If the user wishes to write code to link against any of the libraries in the lib directory, it may be necessary to include these header files.
- lib - is a directory of libraries. After running make install the OS library along with all other COIN-OR libraries are installed in lib.
- ThirdParty - is a directory for third-party software. For example, if AMPL related software such as OSAmplClient is used, then certain AMPL libraries need to be present. This should go into the ASL directory in ThirdParty.

The directories in the OS directory are outlined in Figure 2. The OS directories include the following:

- applications - is a directory that holds the OSAmplClient and OSFileUpload applications in subdirectories called, respectively, amplClient and fileUpload. See Section 9.1 and 14.
- data - is a directory that holds test problems. These test problems are used by the unitTest of the OS Project. Many of these files are also used to illustrate how the OSSolverService works. See Section 7.
- doc - is the directory with documentation, including this *OS User's Manual*.

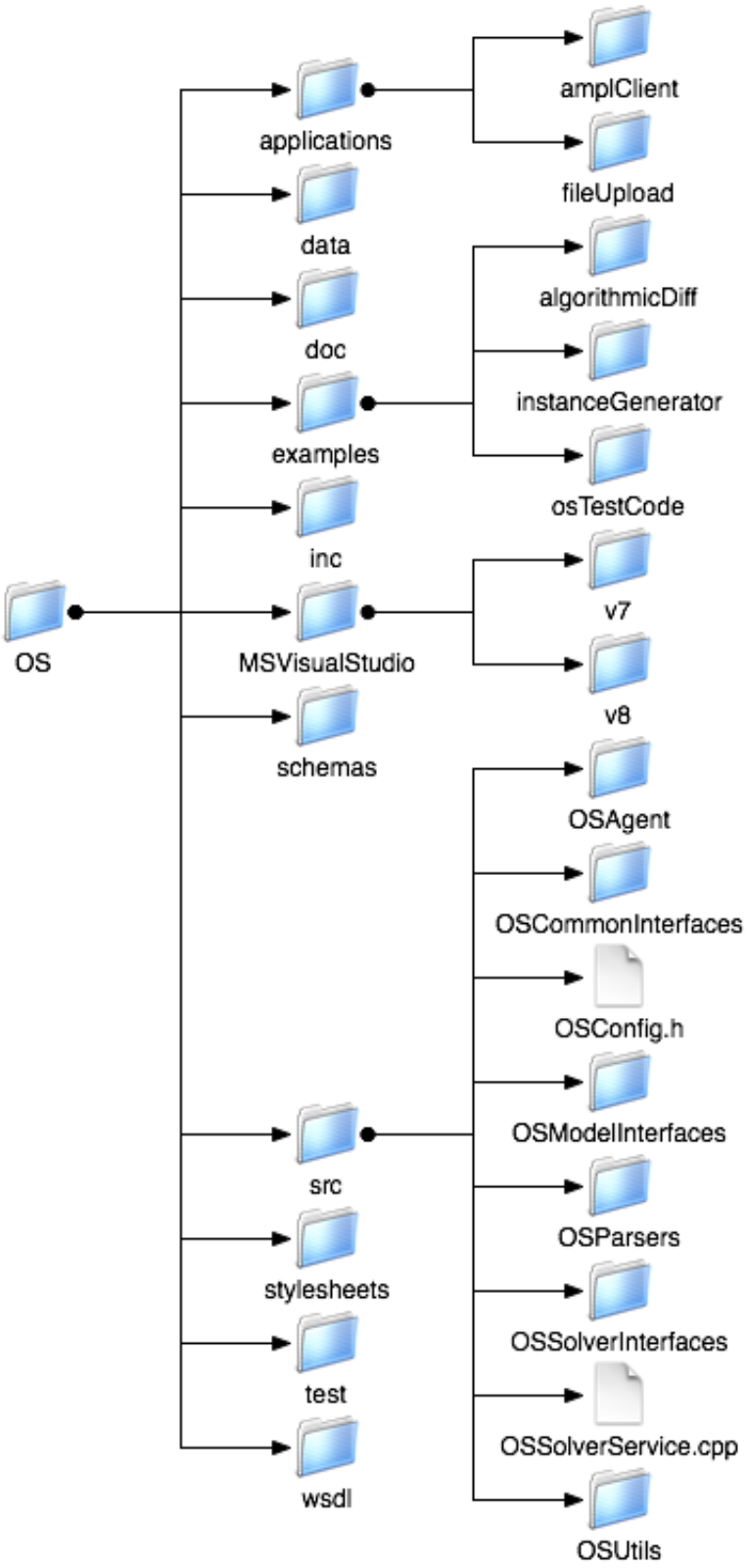

Figure 2: The OS directory.

- `examples` - is a directory with code examples that illustrate various aspects of the OS project. These are described in Section 12.
- `inc` - is the directory with the config˙os.h file which has information about which projects are included in the distribution.
- `m4` - is a directory that contains macro scripts written in the m4 language for auto configuration.
- `MSVisualStudio` - is a directory that contains folders for the solution files for the Microsoft Visual Studio IDE. The subdirectories are organized by the version of Visual Studio. We currently provide solution files for versions 8 and 9.
- `schemas` - is the directory that contains the W3C XSD (see `www.w3.org`) schemas that are behind the OS standards. These are described in more detail in Section 6.
- `src` - is the directory with all of the source code for the OS Library and for the executable `OSSolverService`. The OS Library components are described in Section 10.
- `stylesheets` - this directory contains the XSLT stylesheet that is used to transform the solution instance in OSrL format into HTML so that it can be displayed in a browser.
- `test` - this directory contains the `unitTest`.
- `wsdl` - is a directory of WSDL (Web Services Discovery Language) files. These are used to specify the inputs and outputs for the methods and other invocation details provided by a Web service. The most relevant file for the current version of the OS project is `OShL.wsdl`. This describes the set of inputs and outputs for the methods implemented in the `OSSolverService`. See Section 7.

6 OS Protocols

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. These standards are specified by W3C XSD schemas. The schemas for the OS project are contained in the `schemas` folder under the `OS` root. There are numerous schemas in this directory that are part of the OS standard. For a full description of all the schemas see Ma [4]. We briefly discuss the standards most relevant to the current version of the OS project.

6.1 OSiL (Optimization Services instance Language)

OSiL is an XML-based language for representing instances of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs.

OSiL stores optimization problem instances as XML files. Consider the following problem instance, which is a modification of an example of Rosenbrock [5]:

$$\text{Minimize} \quad (1 - x_0)^2 + 100(x_1 - x_0^2)^2 + 9x_1 \tag{1}$$

$$\text{s.t.} \quad x_0 + 10.5x_0^2 + 11.7x_1^2 + 3x_0x_1 \leq 25 \tag{2}$$

$$\ln(x_0x_1) + 7.5x_0 + 5.25x_1 \geq 10 \tag{3}$$

$$x_0, x_1 \geq 0 \tag{4}$$

There are two continuous variables, x_0 and x_1, in this instance, each with a lower bound of 0. Figure 3 shows how we represent this information in an XML-based OSiL file. Like all XML files, this is a text file that contains both *markup* and *data*. In this case there are two types of markup, *elements* (or *tags*) and *attributes* that describe the elements. Specifically, there are a `<variables>` element and two `<var>` elements. Each `<var>` element has attributes `lb`, `name`, and `type` that describe properties of a decision variable: its lower bound, "name", and domain type (continuous, binary, general integer).

To be useful for communication between solvers and modeling languages, OSiL instance files must conform to a standard. An XML-based representation standard is imposed through the use of a *W3C XML Schema.* The W3C, or World Wide Web Consortium (`www.w3.org`), promotes standards for the evolution of the web and for interoperability between web products. XML Schema (`www.w3.org/XML/Schema`) is one such standard. A schema specifies the elements and attributes that define a specific XML vocabulary. The W3C XML Schema is thus a schema for schemas; it specifies the elements and attributes for a schema that in turn specifies elements and attributes for an XML vocabulary such as OSiL. An XML file that conforms to a schema is called *valid* for that schema.

By analogy to object-oriented programming, a schema is akin to a header file in C++ that defines the members and methods in a class. Just as a class in C++ very explicitly describes member and method names and properties, a schema explicitly describes element and attribute names and properties.

Figure 4 is a piece of our schema for OSiL. In W3C XML Schema jargon, it defines a *complexType,* whose purpose is to specify elements and attributes that are allowed to appear in a valid XML instance file such as the one excerpted in Figure 3. In particular, Figure 4 defines the complexType named `Variables`, which comprises an element named `<var>` and an attribute named `numberOfVariables`. The `numberOfVariables` attribute is of a standard type `positiveInteger`, whereas the `<var>` element is a user-defined complexType named `Variable`. Thus the complexType `Variables` contains a sequence of `<var>` elements that are of complexType `Variable`. OSiL's schema must also provide a specification for the `Variable` complexType, which is shown in Figure 5.

```
<variables numberOfVariables="2">
    <var lb="0" name="x0" type="C"/>
    <var lb="0" name="x1" type="C"/>
</variables>
```

Figure 3: The `<variables>` element for the example (1)–(4).

```
<xs:complexType name="Variables">
    <xs:sequence>
        <xs:element name="var" type="Variable" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="numberOfVariables"
            type="xs:positiveInteger" use="required"/>
</xs:complexType>
```

Figure 4: The `Variables` complexType in the OSiL schema.

In OSiL the linear part of the problem is stored in the `<linearConstraintCoefficients>` element, which stores the coefficient matrix using three arrays as proposed in the earlier LPFML schema [2]. There is a child element of `<linearConstraintCoefficients>` to represent each array: `<value>` for an array of nonzero coefficients, `<rowIdx>` or `<colIdx>` for a corresponding array of row indices or column indices, and `<start>` for an array that indicates where each row or column begins in the previous two arrays. This is shown in Figure 6.

The quadratic part of the problem is represented in Figure 7.

The nonlinear part of the problem is given in Figure 8.

The complete OSiL representation can be found in the Appendix (Section 16.1).

6.2 OSrL (Optimization Services result Language)

OSrL is an XML-based language for representing the solution of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs. An example solution (for the problem given in (1)–(4)) in OSrL format is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type = "text/xsl"
  href = "/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OSX/OS/stylesheets/OSrL.xslt"?>
<osrl xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <generalStatus type="normal"/>
        <serviceName>Solved using a LINDO service</serviceName>
        <instanceName>Modified Rosenbrock</instanceName>
    </general>
    <optimization numberOfSolutions="1" numberOfVariables="2" numberOfConstraints="2"
        numberOfObjectives="1">
        <solution targetObjectiveIdx="-1">
```

```
<xs:complexType name="Variable">
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="init" type="xs:string" use="optional"/>
    <xs:attribute name="type" use="optional" default="C">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="C"/>
                <xs:enumeration value="B"/>
                <xs:enumeration value="I"/>
                <xs:enumeration value="S"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="lb" type="xs:double" use="optional" default="0"/>
    <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>
</xs:complexType>
```

Figure 5: The `Variable` complexType in the OSiL schema.

```
<linearConstraintCoefficients numberOfValues="3">
    <start>
        <el>0</el><el>2</el><el>3</el>
    </start>
    <rowIdx>
        <el>0</el><el>1</el><el>1</el>
    </rowIdx>
    <value>
        <el>1.</el><el>7.5</el><el>5.25</el>
    </value>
</linearConstraintCoefficients>
```

Figure 6: The `<linearConstraintCoefficients>` element for constraints (2) and (3).

```
<quadraticCoefficients numberOfQuadraticTerms="3">
     <qTerm idx="0" idxOne="0" idxTwo="0" coef="10.5"/>
     <qTerm idx="0" idxOne="1" idxTwo="1" coef="11.7"/>
     <qTerm idx="0" idxOne="0" idxTwo="1" coef="3."/>
</quadraticCoefficients>
```

Figure 7: The `<quadraticCoefficients>` element for constraint (2).

```
        <status type="optimal"/>
        <variables>
            <values numberOfVar="2">
                <var idx="0">0.87243</var>
                <var idx="1">0.741417</var>
            </values>
            <other numberOfVar="2" name="reduced costs" description="the variable reduced costs">
                <var idx="0">-4.06909e-08</var>
                <var idx="1">0</var>
            </other>
        </variables>
        <objectives>
            <values numberOfObj="1">
                <obj idx="-1">6.7279</obj>
            </values>
        </objectives>
        <constraints>
            <dualValues numberOfCon="2">
                <con idx="0">0</con>
                <con idx="1">0.766294</con>
            </dualValues>
        </constraints>
    </solution>
</optimization>
```

```
<nl idx="-1">
     <plus>
          <power>
               <minus>
                    <number value="1.0"/>
                    <variable coef="1.0" idx="0"/>
               </minus>
               <number value="2.0"/>
          </power>
          <times>
               <power>
                    <minus>
                         <variable coef="1.0" idx="0"/>
                         <power>
                              <variable coef="1.0" idx="1"/>
                              <number value="2.0"/>
                         </power>
                    </minus>
                    <number value="2.0"/>
               </power>
               <number value="100"/>
          </times>
     </plus>
</nl>
```

Figure 8: The `<nl>` element for the nonlinear part of the objective (1).

6.3 OSoL (Optimization Services option Language)

OSoL is an XML-based language for representing options that get passed to an optimization solver or a hosted optimization solver Web service. It contains both standard options for generic services and extendable option tags for solver-specific directives. Several examples of files in OSoL format are presented in Section 7.4.

6.4 OSnL (Optimization Services nonlinear Language)

The OSnL schema is imported by the OSiL schema and is used to represent the nonlinear part of an optimization instance. This is explained in greater detail in Section 10.2.4. Also refer to Figure 8 for an illustration of elements from the OSnL standard. This figure represents the nonlinear part of the objective in equation (1), that is,

$$(1 - x_0)^2 + 100(x_1 - x_0^2)^2.$$

6.5 OSpL (Optimization Services process Language)

This is a standard used to enquire about dynamic process information that is kept by the Optimization Services registry. The string passed to the `knock` method is in the OSpL format. See the example given in Section 7.4.5.

7 The OSSolverService

The `OSSolverService` is a command line executable designed to pass problem instances in either OSiL, AMPL nl, or MPS format to solvers and get the optimization result back to be displayed either to standard output or a specified browser. The `OSSolverService` can be used to invoke a solver locally or on a remote server. It can work either synchronously or asynchronously. At present six service methods are implemented, `solve`, `send`, `retrieve`, `getJobID`, `knock` and `kill`. These methods are explained in more detail in Section 7.4.

There are two ways to use the `OSSolverService` executable. The first way is to use the interactive shell. The interactive shell is invoked by either double clicking on the icon for the `OSSolverService` executable, or by opening a command window, connecting to the directory holding the executable, and then typing in `OSSolverService` with no arguments. Using the interactive shell is fairly intuitive and we do not discuss in detail. The second way to use the `OSSolverService` executable is to provide arguments at the command line. This is discussed next. The command line arguments are valid for the interactive shell.

7.1 OSSolverService Input Parameters

At present, the `OSSolverService` takes the following parameters. The order of the parameters is irrelevant. Not all the parameters are required.

osil xxx.osil This is the path information and name of the file that contains the optimization instance in OSiL format. It is assumed that this file is available on the machine that is running `OSSolverService`. This option can be omitted, as there are other ways to specify an optimization instance.

osol xxx.osol This is the path information and name of the file that contains the solver options. It is assumed that this file is available on the machine that is running `OSSolverService`. It is not necessary to specify this option.

osrl xxx.osrl This is the path information and name of the file that contains the solver solution. A valid file path must be given on the machine that is running `OSSolverService`. It is not necessary to specify this option. If this option is not specified then the solver solution is displayed to the screen.

osplInput xxx.ospl The name of an input file in the OS Process Language (OSpL); this is used as input to the `knock` method.

osplOutput xxx.ospl The name of an output file in the OS Process Language (OSpL); this is the output string from the `knock` and `kill` method.

serviceLocation url This is the URL of the solver service. It is not required, and if not specified it is assumed that the problem is solved locally.

serviceMethod methodName This is the method on the solver service to be invoked. The options are `solve`, `send`, `kill`, `knock`, `getJobID`, and `retrieve`. The use of these options is illustrated in the examples below. This option is not required, and the default value is `solve`.

mps xxx.mps This is the path information and name of the MPS file if the problem instance is in MPS format. It is assumed that this file is available on the machine that is running `OSSolverService`. The default file format is OSiL so this option is not required.

Table 3: Solver configurations

	binaries (Section 3.1)	UNIX build (Section 4.1)	MSVS build (Section 4.2)
Bonmin	x	x[1]	x[1,2]
Cbc	x	x	x
Clp	x	x	x
Couenne	x	x[1]	—
DyLP	x	x	—
Ipopt	x	x[1]	x[1,2]
SYMPHONY	x	x	x
Vol	x	x	x

Explanations:
[1]Requires third-party software to be downloaded
[2]Requires Fortran compiler (see Section 4.4)

Table 4: Default solvers

Problem type	Default solver
Linear, continuous	Clp
Linear, integer	Cbc
Nonlinear, continuous	Ipopt
Nonlinear, integer	Bonmin

nl xxx.nl This is the path information and name of the AMPL nl file if the problem instance is in AMPL nl format. It is assumed that this file is available on the machine that is running `OSSolverService`. The default file format is OSiL so this option is not required.

solver solverName Possible values of this parameter depend on the installation. The default configurations can be read off from Table 3. Other solvers supported (if the necessary libraries are present) are `cplex` (Cplex through COIN-OR Osi), `glpk` (GLPK through COIN-OR Osi) and `lindo` (LINDO). If no value is specified for this parameter, then a default value is used for the `solve` or `send` service method. The default solver depends on the problem type and can be read off from table 4.

browser browserName This parameter is a path to the browser on the local machine. If this optional parameter is specified then the solver result in OSrL format is transformed using XSLT into HTML and displayed in the browser.

config pathToConfigureFile This optional parameter specifies a path on the local machine to a text file containing values for the input parameters. This is convenient for the user not wishing to constantly retype parameter values.

The input parameters to the `OSSolverService` may be given entirely in the command line or in a configuration file. We first illustrate giving all the parameters in the command line. The following command will invoke the `Clp` solver on the local machine to solve the problem instance `parincLinear.osil`. When invoking the commands below involving `OSSolverService` we assume that 1) the user is connected to the directory where the `OSSolverService` executable is located,

and 2) that `../data/osilFiles` is a valid path to `COIN-OS/data/osilFiles`. If the OS project was built successfully, then there is a copy of `OSSolverService` in `COIN-OS/OS/src`. The user may wish to execute `OSSolverService` from this `src` directory so that all that follows is correct in terms of path definitions.

```
./OSSolverService solver clp osil ../data/osilFiles/parincLinear.osil
```

Alternatively, these parameters can be put into a configuration file. Assume that the configuration file of interest is `testlocalclp.config`. It would contain the two lines of information

```
osil ../data/osilFiles/parincLinear.osil
solver clp
```

Then the command line is

```
./OSSolverService config ../data/configFiles/testlocalclp.config
```

Windows users should **note** that the folder separator is always the forward slash ('/') instead of the customary backslash ('\').

Parameters specified in the configure file are overridden by parameters specified at the command line. This is convenient if a user has a base configure file and wishes to override only a few options. For example,

```
./OSSolverService config ../data/configFiles/testlocalclp.config solver lindo
```

or

```
./OSSolverService solver lindo config ../data/configFiles/testlocalclp.config
```

will result in the LINDO solver being used even though `Clp` is specified in the `testlocalclp` configure file.

Some things to note:

1. The default `serviceMethod` is `solve` if another service method is not specified. The service method cannot be specified in the OSoL options file.
2. If the options `send`, `kill`, `knock`, `getJobID`, or `retrieve` are specified, a `serviceLocation` must be specified.
3. Only the `solve()` method is available for local calls to `OSSolverService`.
4. When using the `send()` or `solve()` methods a problem instance must be specified.

7.2 The Command Line Parser

The top layer of the local OSSolverService is a command line parser that parses the command line and the config file (if one is specified) and passes the information on to a local solver or a remote solver service, depending on whether a `serviceLocation` was specified. If a `serviceLocation` is specified a call is made to a remote solver service, otherwise a local solver is called.

If a local solve is indicated, we pass to a solver in the OSLibrary two things: an OSoL file if one has been specified and a problem instance. The problem instance is the instance in the OSiL file specified by the `osil` option. If there is no OSiL file, then it is the instance specified in the nl

file. If there is no nl file, it is the instance in the mps file. If no OSiL, nl or mps file is specified, an error is thrown.

The OSoL file is simply passed on to the OSLibrary; it is not parsed at this point. The OSoL file elements `<solverToInvoke>` and `instanceLocation` cannot be used for local calls. One can specify which solver to use in the OSLibrary through the `solver` option. If this option is empty, a default solver is selected (see Table 4).

If the `serviceLocation` parameter is used, a call is placed to the remote solver service specified in the `serviceLocation` parameter. Two strings are passed to the remote solver service: a string which is the OSoL file if one has been specified, or the empty string otherwise, and a string containing an instance if one has been specified. The instance can be specified using the `osil`, `-nl`, or `-mps` option. If an OSiL file is specified in the `osil` option, it is used. If there is no OSiL file, then the instance specified in the nl file is used. If there is no nl file, the mps file is used. If no file is given, an empty string is sent.

For remote calls, the solver can only be set in the osol file, using the element `<solverToInvoke>`; the `solver` option has no effect.

7.3 Solving Problems Locally

Generally, when solving a problem locally the user will use the `solve` service method. The `solve` method is invoked synchronously and waits for the solver to return the result. This is illustrated in Figure 9. As illustrated, the `OSSolverService` reads a file on the hard drive with the optimization instance, usually in OSiL format. The optimization instance is parsed into a string which is passed to the `OSLibrary` (see 10), which is linked with various solvers. Similarly an option file in OSoL format is parsed into a string and passed to the `OSLibrary`. *No interpretation of the options is done at this stage*, so that any `<solverToInvoke>` or `<instanceLocation>` directives in the OSoL file will be ignored for local solves. The result of the optimization is passed back to the `OSSolverService` as a string in OSrL format.

Here is an example of using a configure file, `testlocal.config`, to invoke `Ipopt` locally using the `solve` command.

```
osil ../data/osilFiles/parincQuadratic.osil
solver ipopt
serviceMethod solve
browser /Applications/Firefox.app/Contents/MacOS/firefox
osrl /Users/kmartin/temp/test.osrl
```

The first line of `testlocal.config` gives the local location of the OSiL file, `parincQuadratic.osil`, that contains the problem instance. The second parameter, `solver ipopt`, is the solver to be invoked, in this case COIN-OR Ipopt. The third parameter `serviceMethod solve` is not really needed, since the default solver service is `solve`. It is included only for illustration. The fourth parameter is the location of the browser on the local machine. The fifth parameter is `osrl`. The value of this parameter, `/Users/kmartin/temp/test.osrl`, specifies the location on the local machine where the OSrL result file will get written.

Parameters may also be contained in an XML-file in OSoL format. In the configuration file `testlocalosol.config` we illustrate specifying the instance location in an OSoL file.

```
osol ../data/osolFiles/demo.osol
solver clp
```

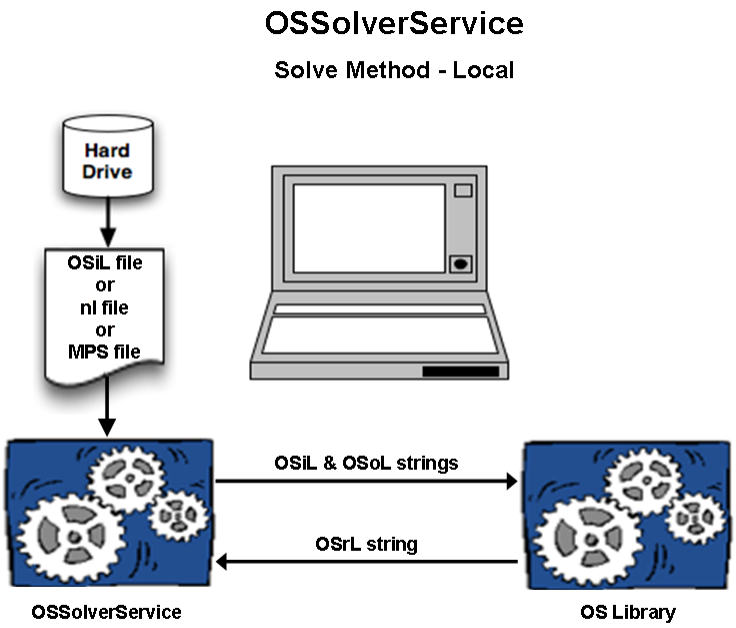

Figure 9: A local call to `solve`.

The file `demo.osol` is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <instanceLocation locationType="local">
            ../data/osilFiles/parincLinear.osil
        </instanceLocation>
    </general>
</osol>
```

7.4 Solving Problems Remotely with Web Services

In many cases the client machine may be a "weak client" and using a more powerful machine to solve a hard optimization instance is required. Indeed, one of the major purposes of Optimization Services is to facilitate optimization in a distributed environment. We now provide examples that illustrate using the `OSSolverService` executable to call a remote solver service. By remote solver service we mean a solver service that is called using Web Services. The OS implementation of the solver service uses Apache Tomcat. See `tomcat.apache.org`. The Web Service running on the server is a Java program based on Apache Axis. See `ws.apache.org/axis`. This is described in greater detail in Section 8. This Web Service is called `OSSolverService.jws`. It is not necessary to use the Tomcat/Axis combination.

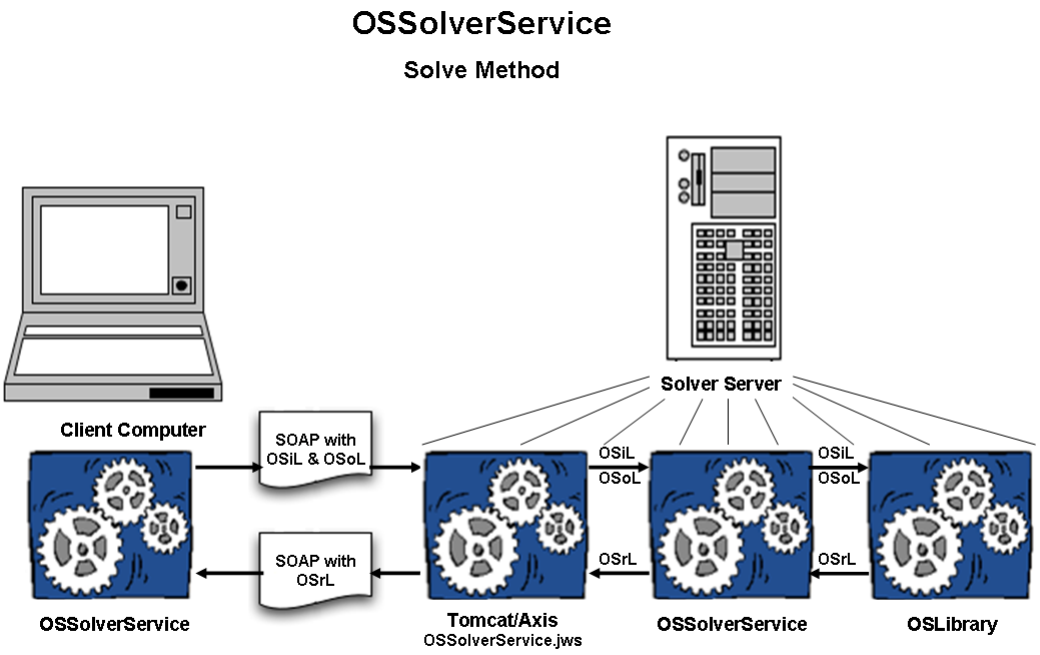

Figure 10: A remote call to `solve`.

See Figure 10 for an illustration of this process. The client machine uses `OSSolverService` executable to call one of the six service methods, e.g., `solve`. The information such as the problem instance in OSiL format and solver options in OSoL format are packaged into a SOAP envelope and sent to the server. The server is running the Java Web Service `OSSolverService.jws`. This Java program running in the Tomcat Java Servlet container implements the six service methods. If a `solve` or `send` request is sent to the server from the client, an optimization problem must be solved. The Java solver service solves the optimization instance by calling the `OSSolverService` on the server. So there is an `OSSolverService` on the client that calls the Web Service `OSSolverService.jws` that in turn calls the executable `OSSolverService` on the server. The Java solver service passes options to the local `OSSolverService` in form of two strings, an osil string representing the instance and an osol string representing the options (if any).

For remote calls the instance location can be specified either as a command parameter (on the command line or in a config file) or through the `<instanceLocation>` element in the OSoL options file. OSiL files specified in the `<instanceLocation>` element must be converted to an osil string by the solver service. If two instance files are specified in this way — one through the local command interface, the other in an options file — the solver service is free to pick which one to choose.

In the following sections we illustrate each of the six service methods.

7.4.1 The `solve` Service Method

First we illustrate a simple call to `OSSolverService.jws`. The call on the client machine is

```
./OSSolverService config ../data/configFiles/testremote.config
```

where the `testremote.config` file is

```
osil ../data/osilFiles/parincLinear.osil
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
```

No solver is specified and by default the `Clp` solver is used by the `OSSolverService`, since the problem is a continuous linar program. If, for example, the user wished to solve the problem with the `SYMPHONY` solver then this is accomplished either by using the `solver` option on the command line

```
./OSSolverService config ../data/configFiles/testremote.config solver symphony
```

or by adding the line

```
 solver symphony
```

to the `testremote.config` file.

Next we illustrate a call to the remote SolverService and specify an OSiL instance that is actually residing on the remote machine that is hosting the `OSSolverService` and not on the client machine.

```
./OSSolverService osol ../data/osolFiles/remoteSolve1.osol
     serviceLocation  http://kipp.chicagobooth.edu/os/OSSolverService.jws
```

where the `remoteSolve1.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <instanceLocation locationType="local">c:\parincLinear.osil</instanceLocation>
        <contact transportType="smtp">kipp.martin@chicagogsb.edu</contact>
        <solverToInvoke>ipopt</solverToInvoke>
    </general>
</osol>
```

If we were to change the `locationType` attribute in the `<instanceLocation>` element to `http` then we could specify the instance location on yet another machine. This is illustrated below for `remoteSolve2.osol`. The scenario is depicted in Figure 11. The OSiL string passed from the client to the solver service is empty. However, the OSoL element `<instanceLocation>` has an attribute `locationType` equal to `http`. In this case, the text of the `<instanceLoction>` element contains the URL of a third machine which has the problem instance `parincLinear.osil`. The solver service will contact the machine with URL `http://www.coin-or.org/OS/parincLinear.osil` and download this test problem. So the `OSSolverService` is running on the server `kipp.chicagobooth.edu` which contacts the server `www.coin-or.org` for the model instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
```

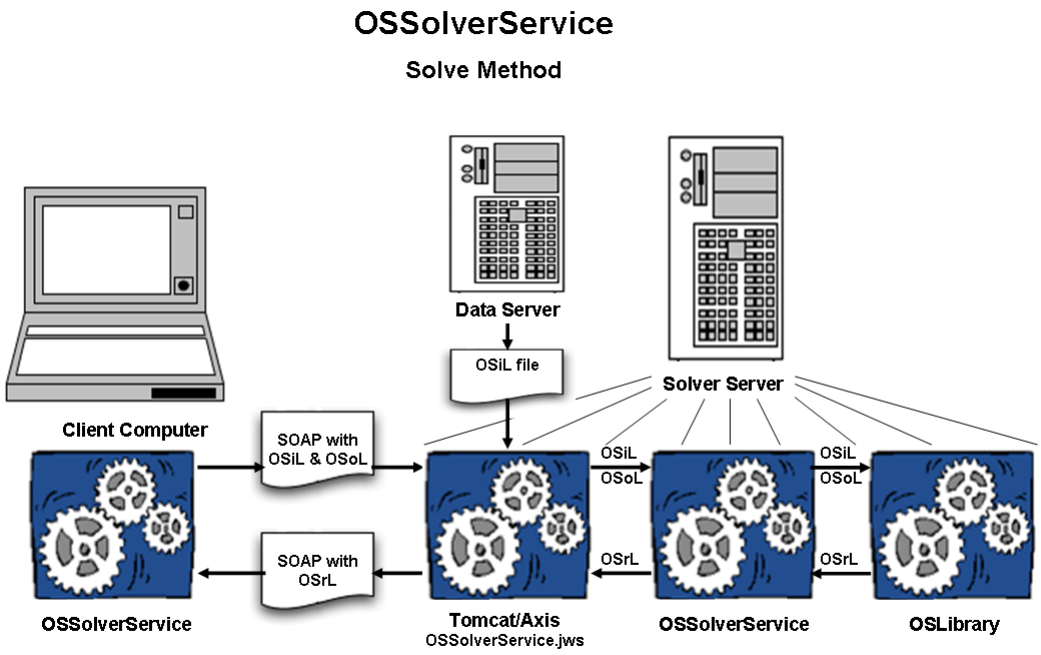

Figure 11: Downloading the instance from a remote source.

```
        <instanceLocation locationType="http">
            http://www.coin-or.org/OS/parincLinear.osil
        </instanceLocation>
        <solverToInvoke>ipopt</solverToInvoke>
    </general>
</osol>
```

Note: The `solve` method communicates synchronously with the remote solver service and once started, these jobs cannot be killed. This may not be desirable for large problems when the user does not want to wait for a response or when there is a possibility for the solver to enter an infinite loop. The `send` service method should be used when asynchronous communication is desired.

7.4.2 The `send` Service Method

When the `solve` service method is used, then the `OSSolverService` does not finish execution until the solution is returned from the remote solver service. When the `send` method is used, the instance is communicated to the remote service and the `OSSolverService` terminates after submission. An example of this is

```
./OSSolverService config ../data/configFiles/testremoteSend.config
```

where the `testremoteSend.config` file is

```
-nl ../data/amplFiles/hs71.nl
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
serviceMethod send
```

In this example the COIN-OR `Ipopt` solver is specified. The input file `hs71.nl` is in AMPL nl format. Before sending this to the remote solver service the `OSSolverService` executable converts the nl format into the OSiL XML format and packages this into the SOAP envelope used by Web Services.

Since the `send` method involves asynchronous communication the remote solver service must keep track of jobs. The `send` method requires a `JobID`. In the above example no `JobID` was specified. When no `JobID` is specified the `OSSolverService` method first invokes the `getJobID` service method to get a `JobID`, puts this information into an OSoL file it creates, and sends the information to the server. More information on the `getJobID` service method is provided in Section 7.4.4. The `OSSolverService` prints the OSoL file to standard output before termination. This is illustrated below,

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <jobID>
            gsbrkm4__127.0.0.1__2007-06-16T15.46.46.075-05.00149771253
        </jobID>
        <solverToInvoke>ipopt</solverToInvoke>
    </general>
</osol>
```

The `JobID` is one that is randomly generated by the server and passed back to the `OSSolverService`. The user can also provide a `JobID` in their OSoL file. For example, below is a user-provided OSoL file that could be specified in a configuration file or on the command line.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <jobID>123456abcd</jobID>
        <solverToInvoke>ipopt</solverToInvoke>
    </general>
</osol>
```

The same `JobID` cannot be used twice, so if `123456abcd` was used earlier, the result of `send` will be `false`.

In order to be of any use, it is necessary to get the result of the optimization. This is described in Section 7.4.3. Before proceeding to this section, we describe two ways for knowing when the optimization is complete. One feature of the standard OS remote SolverService is the ability to send an email when the job is complete. Below is an example of the `OSoL` that uses the email feature.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <jobID>123456abcd</jobID>
        <contact transportType="smtp">
            kipp.martin@chicagogsb.edu
        </contact>
        <solverToInvoke>ipopt</solverToInvoke>
    </general>
</osol>
```

The remote Solver Service will send an email to the above address when the job is complete. A second option for knowing when a job is complete is to use the `knock` method. (See Section 7.4.5.)

Note that in all of these examples we provided a value for the `<solverToInvoke>` element. A default solver is used (see Table 4 if another solver is not specified.

7.4.3 The `retrieve` Service Method

The `retrieve` method is used to get information about the instance solution. This method has a single string argument which is an OSoL instance. Here is an example of using the `retrieve` method with `OSSolverService`.

```
./OSSolverService config ../data/configFiles/testremoteRetrieve.config
```

The `testremoteRetrieve.config` file is

```
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
osol ../data/osolFiles/retrieve.osol
serviceMethod retrieve
osrl /home/kmartin/temp/test.osrl
```

and the `retrieve.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

The OSoL file `retrieve.osol` contains a tag `<jobID>` that is communicated to the remote service. The remote service locates the result and returns it as a string. The `<jobID>` should reflect a `<jobID>` that was previously submitted using a `send()` command. The result is returned as a string in OSrL format. The user must modify the line

```
osrl /home/kmartin/temp/test.osrl
```

to reflect a valid path for their own machine. (It is also possible to delete the line in which case the result will be displayed on the screen instead of being saved to the file indicated in the `osrl` option.)

7.4.4 The `getJobID` Service Method

Before submitting a job with the `send` method a `JobID` is required. The `OSSolverService` can get a `JobID` with the following options.

```
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
serviceMethod getJobID
```

Note that no OSoL input file is specified. In this case, the `OSSolverService` sends an empty string. A string is returned with the `JobID`. This `JobID` is then put into a `<jobID>` element in an OSoL string that would be used by the `send` method.

7.4.5 The `knock` Service Method

The OSSolverService terminates after executing the `send` method. Therefore, it is necessary to know when the job is completed on the remote server. One way is to include an email address in the `<contact>` element with the attribute `transportType` set to `smtp`. This was illustrated in Section 7.4.1. A second way to check on the status of a job is to use the `knock` service method. For example, assume a user wants to know if the job with `JobID 123456abcd` is complete. A user would make the request

```
./OSSolverService config ../data/configFiles/testRemoteKnock.config
```

where the `testRemoteKnock.config` file is

```
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
osplInput ../data/osolFiles/demo.ospl
osol ../data/osolFiles/retrieve.osol
serviceMethod knock
```

the `demo.ospl` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<ospl xmlns="os.optimizationservices.org">
    <processHeader>
        <request action="getAll"/>
    </processHeader>
    <processData/>
</ospl>
```

and the `retrieve.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
```

```
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

The result of this request is a string in OSpL format, with the data contained in its `processData` section. The result is displayed on the screen; if the user desires it to be redirected to a file, a command should be added to the `testRemoteKnock.config` file with a valid path name on the local system, e.g.,

```
osplOutput ./result.ospl
```

Part of the return format is illustrated below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ospl xmlns="os.optimizationservices.org">
  <processHeader>
    <serviceURI>http://localhost:8080/os/ossolver/CGSolverService.jws</serviceURI>
    <serviceName>CGSolverService</serviceName>
    <time>2006-05-10T15:49:26.7509413-05:00</time>
  <processHeader>
  <processData>
     <statistics>
        <currentState>idle</currentState>
        <availableDiskSpace>23440343040</availableDiskSpace>
        <availableMemory>70128</availableMemory>
        <currentJobCount>0</currentJobCount>
        <totalJobsSoFar>1</totalJobsSoFar>
        <timeServiceStarted>2006-05-10T10:49:24.9700000-05:00</timeServiceStarted>
        <serviceUtilization>0.1</serviceUtilization>
        <jobs>
        <job jobID="123456abcd">
            <state>finished</state>
            <serviceURI>http://kipp.chicagobooth.edu/ipopt/IPOPTSolverService.jws</serviceURI>
            <submitTime>2007-06-16T14:57:36.678-05:00</submitTime>
            <startTime>2007-06-16T14:57:36.678-05:00</startTime>
            <endTime>2007-06-16T14:57:39.404-05:00</endTime>
            <duration>2.726</duration>
          </job>
        </jobs>
     </statistics>
  </processData>
</ospl>
```

Notice that the `<state>` element in `<job jobID="123456abcd">` indicates that the job is finished.

When making a `knock` request, the OSoL string can be empty. In this example, if the OSoL string had been empty the status of all jobs kept in the file ospl.xml is reported. In our default solver service implementation, there is a configuration file `OSParameter` that has a parameter `MAX_JOBIDS_TO_KEEP` . The current default setting is 100. In a large-scale or commercial implementation it might be wise to keep problem results and statistics in a database. Also, there are

values other than `getAll` (i.e., get all process information related to the jobs) for the OSpL `action` attribute in the `<request>` tag. For example, the `action` can be set to a value of `ping` if the user just wants to check if the remote solver service is up and running. For details, check the OSpL schema.

7.4.6 The `kill` Service Method

If the user submits a job that is taking too long or is a mistake, it is possible to kill the job on the remote server using the `kill` service method. For example, to kill job `123456abcd`, at the command line type

```
./OSSolverService config  ../data/configFiles/kill.config
```

where the configure file `kill.config` is

```
osol ../data/osolFiles/kill.osol
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
serviceMethod kill
```

and the `kill.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

The result is returned in OSpL format.

7.4.7 Summary and description of the API

The six service methods just described are also available as callable routines. Below is a summary of the inputs and outputs of the six methods. See also Figure 12. A test program illustrating the use of the methods is described in Section 12.8.

- `solve( osil, osol ):`
  - Inputs: a string with the instance in OSiL format and an optional string with the solver options in OSoL format
  - Returns: a string with the solver solution in OSrL format
  - Synchronous call, blocking request/response
- `send( osil, osol ):`
  - Inputs: a string with the instance in OSiL format and a string with the solver options in OSoL format (same as in `solve`)
  - Returns: a boolean, true if the problem was successfully submitted, false otherwise

  - Has the same signature as `solve`
  - Asynchronous (server side), non-blocking call
  - The `osol` string should have a `JobID` in the `<jobID>` element

- `getJobID( osol ):`
  - Inputs: a string with the solver options in OSoL format (in this case, the string may be empty because no options are required to get the JobID)
  - Returns: a string which is the unique job id generated by the solver service
  - Used to maintain session and state on a distributed system

- `knock( ospl, osol ):`
  - Inputs: a string in OSpL format and an optional string with the solver options in OSoL format
  - Returns: process and job status information from the remote server in OSpL format

- `retrieve( osol ):`
  - Inputs: a string with the solver options in OSoL format
  - Returns: a string with the solver solution in OSrL format
  - The `osol` string should have a `JobID` in the `<jobID>` element

- `kill( osol ):`
  - Inputs: a string with the solver options in OSoL format
  - Returns: process and job status information from the remote server in OSpL format
  - Critical in long running optimization jobs

7.5 Passing Options to Solvers

The OSoL (Optimization Services option Language) protocol is used to pass options to solvers. When using the `OSSolverService` executable this will typically be done through an OSoL XML file by specifying the `osol` option followed by the location of the file. However, it is also possible to write a custom application that links to the OS library and to build an OSOption object in memory and then pass this to a solver. We next describe the feature of the OSoL protocol that will be the most useful to the typical user.

In the OSoL protocol there is an element `<solverOptions>` that can have any number of `<solverOption>` children. (See the file `parsertest.osol` in OS/data/osolFiles.) Each `<solverOption>` child can have six attributes, all of which except one are optional. These attributes are:

- **name:** this is the only required attribute and is the option name. It should be unique.
- **value:** the value of the option.
- **solver:** the name of the solver associated with the option.

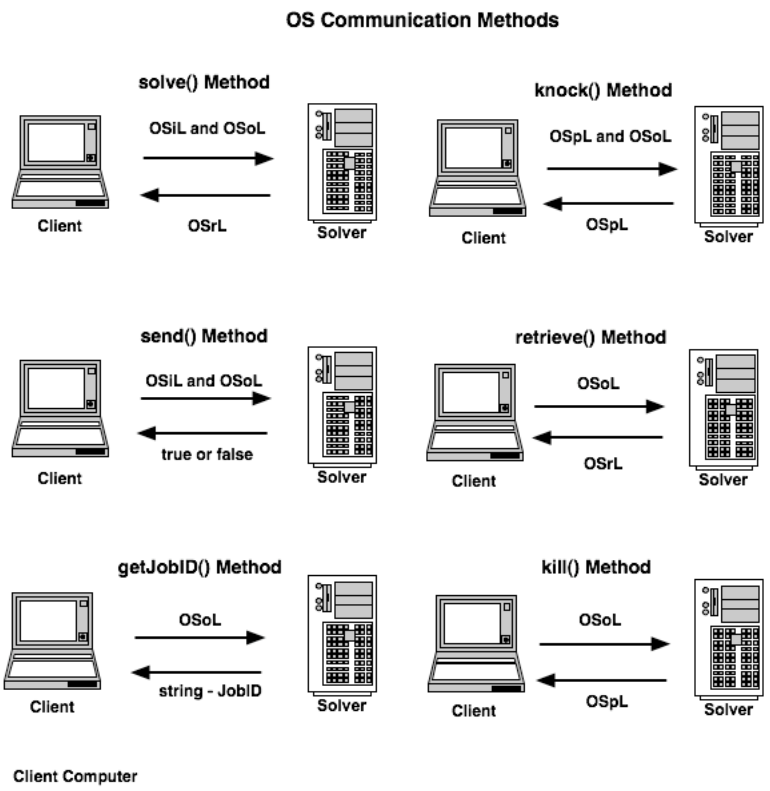

Figure 12: The OS Communication Methods

- **type:** this will usually be a data type (such as integer, string, double, etc.) but this is not necessary.

- **category:** the same solver option may apply to multiple categories so it may be necessary to specify a category for solver. For example, in LINDO an option can apply to a specific model or to every model in an environment. Hence we might have

```
<solverOption name="LS_IPARAM_LP_PRINTLEVEL"
   solver="lindo" category="model"  type="integer" value="0"/>
<solverOption name="LS_IPARAM_LP_PRINTLEVEL"
   solver="lindo" category="environment" type="integer" value="1"/>
```

  where we specify the print level for a specific model or the entire environment. The category attribute should be separated by a colon (':') if there is more than one category or additional subcategories, as in the following hypothetical example.

```
<solverOption name="hypothetical"
   solver="SOLVER" category="cat1:subcat2:subsubcat3"
      type="string" value="illustration"/>
```

- **description:** a description of the option; typically this would not get passed to the solver.

As of trunk version 2164 the reading of an OSoL file is implemented in the OSCoinSolver, OSBonmin and OSIpopt solver interfaces. The OSBonmin, and OSIpopt solvers have particularly easy interfaces. They have methods for integer, string, and numeric data types and then take options in form of (name, value) pairs. Below is an example of options for Ipopt.

```
<solverOption name="mu_strategy" solver="ipopt"
     type="string" value="adaptive"/>
<solverOption name="tol" solver="ipopt"
     type="numeric" value="1.e-9"/>
<solverOption name="print_level" solver="ipopt"
     type="integer" value="5"/>
<solverOption name="max_iter" solver="ipopt"
     type="integer" value="2000"/>
```

We have also implemented the OSOption class for the OSCoinSolver interface. This can be done in two ways. First, options can be set through the Osi Solver interface (the OSCoinSolver interface wraps around the Osi Solver interface). We have implemented all of the options listed in OsiSolverParameters.hpp in Osi trunk version 1316. In the Osi solver interface, in addition to string, double, and integer types there is a type called HintParam and a type called OsiHintParam. The value of the OsiHintParam is an OsiHintStrength type, which may be confusing. For example, to have the following Osi method called

```
setHintParam(OsiDoReducePrint, true, hintStrength);
```

the user should set the following <solverOption> tags:

```
<solverOption name="OsiDoReducePrint" solver="osi"
    type="OsiHintParam"  value="true" />
<solverOption name="OsiHintIgnore" solver="osi"
     type="OsiHintStrength" />
```

There should be only one <solverOption> with type OsiHintStrength and if there are more than one in the OSoL file (string) the last one is the one implemented.

In addition to setting options using the Osi Solver interface, it is possible to pass options directly to the Cbc solver. By default the following options are sent to the Cbc solver,

```
-log=0  -solve
```

The option -log=0 will keep the branch-and-bound output to a minimum. Default options are overridden by putting into the OSoL file at least one <solverOption> tag with the solver attribute set to cbc. For example, the following sequence of options will limit the search to 100 nodes, cut generation turned off.

```
<solverOption name="maxN" solver="cbc" value="100" />
<solverOption name="cuts" solver="cbc" value="off" />
<solverOption name="solve" solver="cbc"  />
```

Any option that Cbc accepts at the command line can be put into a <solverOption> tag. We list those below.

```
Double parameters:
  dualB(ound)  dualT(olerance)  primalT(olerance)  primalW(eight)
Branch and Cut double parameters:
  allow(ableGap)  cuto(ff)  inc(rement)  inf(easibilityWeight)  integerT(olerance)
  preT(olerance)  ratio(Gap)  sec(onds)
Integer parameters:
  cpp(Generate)  force(Solution)  idiot(Crash)  maxF(actor)  maxIt(erations)
  output(Format)  slog(Level)  sprint(Crash)
Branch and Cut integer parameters:
  cutD(epth)  log(Level)  maxN(odes)  maxS(olutions)  passC(uts)
  passF(easibilityPump)  passT(reeCuts)  pumpT(une)  strat(egy)  strong(Branching)
  trust(PseudoCosts)
Keyword parameters:
  chol(esky)  crash  cross(over)  direction  dualP(ivot)
  error(sAllowed)  keepN(ames)  mess(ages)  perturb(ation)  presolve
  primalP(ivot)  printi(ngOptions)  scal(ing)
Branch and Cut keyword parameters:
  clique(Cuts)  combine(Solutions)  cost(Strategy)  cuts(OnOff)  Dins
  DivingS(ome)  DivingC(oefficient)  DivingF(ractional)  DivingG(uided)  DivingL(ineSearch)
  DivingP(seudoCost)  DivingV(ectorLength)  feas(ibilityPump)  flow(CoverCuts)  gomory(Cuts)
  greedy(Heuristic)  heur(isticsOnOff)  knapsack(Cuts)  lift(AndProjectCuts)  local(TreeSearch)
  mixed(IntegerRoundingCuts)  node(Strategy)  pivot(AndFix)  preprocess  probing(Cuts)
  rand(omizedRounding)  reduce(AndSplitCuts)  residual(CapacityCuts)  Rens  Rins
  round(ingHeuristic)  sos(Options)  two(MirCuts)
Actions or string parameters:
  allS(lack)  barr(ier)  basisI(n)  basisO(ut)  directory
  dirSample  dirNetlib  dirMiplib  dualS(implex)  either(Simplex)
  end  exit  export  help  import
  initialS(olve)  max(imize)  min(imize)  netlib  netlibD(ual)
  netlibP(rimal)  netlibT(une)  primalS(implex)  printM(ask)  quit
  restore(Model)  saveM(odel)  saveS(olution)  solu(tion)  stat(istics)
  stop  unitTest  userClp
Branch and Cut actions:
  branch(AndCut)  doH(euristic)  miplib  prio(rityIn)  solv(e)
  strengthen  userCbc
```

The user may also wish to specify an initial starting solution. This is particularly useful with interior point methods. This is accomplished by using the `<initialVariableValues>` tag. Below we illustrate how to set the initial values for variables with an index of 0, 1, and 3.

```
<initialVariableValues numberOfVar="3">
   <var idx="0" value="1"/>
   <var idx="1" value="4.742999643577776" />
   <var idx="3" value="1.379408293215363"/>
</initialVariableValues>
```

As of trunk version 2164 the initial values for variables can be passed to the `Bonmin` and `Ipopt` solvers.

When implementing solver options in-memory, the typical calling sequence is:

```
solver->buildSolverInstance();
solver->setSolverOptions();
solver->solve();
```

8 Setting up a Solver Service with Apache Tomcat

This section explains how to download and use the Java implementation of the remote solver service described in Section 7.4. The server side of the Java distribution is based on the Tomcat 6.0 implementation. In order to build an OS Solver Service, the user should do an svn checkout:

```
svn co https://projects.coin-or.org/svn/OS/branches/OSjava OSjava
```

The `OSjava` folder contains the file `INSTALL.txt`. Please follow the instructions in `INSTALL.txt` under the heading:

```
== Install An OS Web Server==
```

Installing the OS Web Server based on the instructions in `INSTALL.txt` assumes that the user has installed:

- Eclipse IDE. See `http://www.eclipse.org/downloads/`. At this link we recommend that the user get `Eclipse Classic`.
- An `OSSolverService` that is compatible with the server platform. The `OSSolverService` executable for several different platforms is available at `http://www.coin-or.org/download/binary/OS/OSSolverService/`. The user can also build the executable as described in this Manual. See Section 4.
- Tomcat 6.0. See `http://tomcat.apache.org/`.

After the final installation is complete on the server we recommend testing by doing someting like the following. On a client machine, create the file `testremote.config` with the following lines of text

```
serviceLocation http://***.***.***.***:8080/OSServer/services/OSSolverService
osil /parincLinear.osil
```

where `***.***.***.***` is the IP address of the Tomcat server machine. Then, assuming the files `testremote.config` and `parincLinear.osil` are in the same directory on the client machine as the `OSSolverService` execute:

```
./OSSolverService config testremote.config
```

You should get back an OSrL result printed to the screen.

In the following discussion, we assume that the root folder for Tomcat running on the server is named `tomcat`.

If you already have a Tomcat 6.0 server with Axis installed, and have created an `OSServer.war` file based on `INSTALL.txt`, do the following:

1. copy the file `OSServer.war` into the Tomcat `tomcat/webapps` directory.
2. Stop and start Tomcat.

In the directory,

```
tomcat/webapps/OSServer/WEB-INF/code/OSConfig
```

there is a configuration file `OSParameter.xml` that can be modified to fit individual user needs. You can configure such parameters as service name, service URL/URI. Refer to the xml file for more detail. Descriptions for all the parameters are within the file itself.

Below is a summary of the common and important directories and files you may want to know.

- `tomcat/webapps/OSServer/`

  contains the OS Solver Service Web application. All directories and files outside of this folder are Tomcat server related.

- `tomcat/webapps/OSServer/WEB-INF`

  contains private and important configuration, library, class and executable files to run the Optimization Service.

- `tomcat/webapps/OSServer/WEB-INF/code/OSConfig`

  contains configuration files for Optimization Services, such as the `OSParameter.xml` file.

- `tomcat/webapps/OSServer/WEB-INF/code/temp`

  contains temporarily saved files such as submitted OSiL/OSoL input files, and OSrL output files. This folder can get bigger as the service starts to run more jobs. For maintenance purpose, you may want to keep an eye on it.

- `tomcat/webapps/OSServer/WEB-INF/code/log`

  contains log files from the running services in the current Web application.

- `tomcat/webapps/OSServer/WEB-INF/classes`

  contains solver binaries that actually carry out the optimization process.

- `tomcat/webapps/OSServer/WEB-INF/code/backup`

  contains backup files from some of the above directories. This folder can get bigger as the service starts to run more jobs.

- `tomcat/webapps/OSServer/WEB-INF/classes`

  contains class files to run the Optimization Services.

- `tomcat/webapps/OSServer/WEB-INF/lib`

  contains library files needed by the Optimization Services.

- `tomcat/conf`

  contains configuration files for the Tomcat server, such as http server port.

- `tomcat/bin`

  contains executables and scripts to start and shut down the Tomcat server.

9 OS Support for Modeling Languages, Spreadsheets and Numerical Computing Software

Algebraic modeling languages can be used to generate model instances as input to an OS compliant solver. We describe two such hook-ups, `OSAmplClient` for AMPL, and `CoinOS` for GAMS (version 23.3 and above).

9.1 AMPL Client: Hooking AMPL to Solvers

It is possible to call all of the COIN-OR solvers listed in Table 3 (p.37) directly from the AMPL (see `http://www.ampl.com`) modeling language. In this discussion we assume the user has already obtained and installed AMPL. Both the binary download described in Section 3.1 and the unix and Windows builds (Section 4.1 and 4.2, respectively) contain an executable, `OSAmplClient.exe`, that is linked to all of the COIN-OR solvers listed in Table 3. From the perspective of AMPL, the `OSAmplClient` acts like an AMPL "solver". The `OSAmplClient.exe` can be used to solve problems either locally or remotely.

9.1.1 Using OSAmplClient for a Local Solver

In the following discussion we assume that the AMPL executable `ampl.exe`, the `OSAmplClient`, and the test problem `eastborne.mod` are all in the same directory.

The problem instance `eastborne.mod` is an AMPL model file included in the OS distribution in the `amplFiles` directory. To solve this problem locally by calling `OSAmplClient.exe` from AMPL, first start AMPL and then open the `eastborne.mod` file inside AMPL. The test model `eastborne.mod` is a linear integer program.

```
# take in sample integer linear problem
# assume the problem is in the AMPL directory
model eastborne.mod;
```

The next step is to tell AMPL that the solver it is going to use is `OSAmplClient.exe`. Do this by issuing the following command inside AMPL.

```
# tell AMPL that the solver is OSAmplClient
option solver OSAmplClient;
```

It is not necessary to provide the `OSAmplclient.exe` solver with any options. You can just issue the `solve` command in AMPL as illustrated below.

```
# solve the problem
solve;
```

Of the six methods described in Section 7 only the `solve` method has been implemented to date.

If no options are specified, the default solver is used, depending on the problem characteristics (see Table 4). If you wish to specify a specific solver, use the `solver` option. For example, since the test problem `eastborne.mod` is a linear integer program, `Cbc` is used by default. If you want to instead use `SYMPHONY`, then you would pass a `solver` option to the `OSAmplclient.exe` solver as follows.

```
# now tell OSAmplClient to use SYMPHONY instead of Cbc
option OSAmplClient_options "solver symphony";
```

Valid values for the `solver` option are `bonmin`, `cbc`, `clp`, `couenne`, `dylp`, `symphony`, and `vol`. The solver name in the `solver` option is case insensitive.

9.1.2 Using OSAmplClient to Invoke the COIN-OR Solver Server

Next, assume that you have a large problem you want to solve on the remote solver. It is necessary to specify the location of the server solver as an option to OSAmplClient. The `serviceLocation` option is used to specify the location of a solver server. In this case, the string of options for `OSAmplClient_options` is:

```
serviceLocation  http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
```

This string is used to replace the string '`solver symphony`' in the previous example. We will omit the other parts (i.e., the AMPL instruction

```
option OSAmplClient_options
```

the double quotes and the trailing semicolon) in this and the remaining examples.

This option will send the problem to the solver server at location `http://webdss.ise.ufl.edu`.

Each call

```
option OSAmplClient_options
```

is memoryless. That is, the options set in the last call will overwrite any options set in previous calls and cause them to be discarded.

For instance, the sequence of option calls

```
option OSAmplClient_options "solver symphony";
option OSAmplClient_options "serviceLocation
    http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService";
solve;
```

will result in the default solver being called.

Finally, the user may wish to pass options to the individual solver. This is done by providing an options file. A sample options file, `solveroptions.osol` is provided with this distribution. The name of the options file is the value of the `osol` option. The string of options to `OSAmplClient_options` is now

```
serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
osol solveroptions.osol
```

This `solveroptions.osol` file contains four solver options; two for `Cbc`, one for `Ipopt`, and one for `SYMPHONY`. You can have any number of options. Note the format for specifying an option:

```
<solverOption name="maxN" solver="cbc" value="5" />
```

The attribute `name` specifies that the option name is `maxN` which is the maximum number of nodes allowed in the branch-and-bound tree, the `solver` attribute specifies the name of the solver that the option should be applied to, and the `value` attribute specifies the value of the option. As a second example, consider the specification

```
<solverOption name="max_iter" solver="ipopt" type="integer" value="2000"/>
```

In this example we are specifying an iteration limit for `Ipopt`. Note the additional attribute `type` that has value `integer`. The Ipopt solver requires specifying the data type (string, integer, or numeric) for its options. Different solvers have different options, and we recommend that the user look at the documentation for the solver of interest in order to see which options are available. A good summary of options for COIN-OR solvers is `http://www.coin-or.org/GAMSlinks/gamscoin.pdf`.

If you examine the file `solveroptions.osol` you will see that there is an XML tag with the name `<solverToInvoke>` and that the solver given is `symphony`. This has no effect on a local solve. However, if this option file is paired with

```
serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
osol solveroptions.osol
```

then in our reference implementation the remote solver service will parse the file `solveroptions.osol`, find the `<solverToInvoke>` tag and then pass the `symphony` solver option to the `OSSolverService` on the remote server.

9.1.3 AMPL Summary

1. Tell AMPL to use the OSAmplClient as the solver:

   ```
   option solver OSAmplClient;
   ```

2. Specify options to the OSAmplclient solver by using the AMPL command `OSAmplClient_options`.

3. There are three possible options to specify:

   - the name of the solver using the `solver` option; valid values for this option are `clp`, `cbc`, `dylp`, `ipopt`, `bonmin`, `couenne`, `symphony`, and `vol`;
   - the location of the remote server using the `serviceLocation` option;
   - the location of the options file using the `osol` option.

   These three options behave *exactly like* the `solver`, `serviceLocation`, and `osol` options used by the `OSSolverService` described in Section 7.2.

4. If no options are specified using `OSAmplClient_options`, the default solver is used. (For details see Table 4). All solvers are invoked locally.

5. The options given to `OSAmplClient_options` can be given in any order.

6. A remote solver is called if and only if the `serviceLocation` option is specified.

9.2 GAMS and Optimization Services

This section pertains to GAMS version 23.3 (and above) that now includes support for OS. Here we describe the GAMS implementation of Optimization Services. We assume that the user has installed GAMS.

There are two ways to access an OS Solver Service from GAMS, on the local machine or on a remote server. The difference between the two approaches is explained in the next two sections.

9.2.1 Using GAMS to Invoke the Local OS Solver Service `CoinOS`

In GAMS, OS is implemented through the `CoinOS` solver that is packaged with GAMS. The GAMS `CoinOS` solver is really a *solver interface* and is linked through the OS library to the following COIN-OR solvers: `Bonmin`, `Cbc`, `Clp`, `Glpk`, and `Ipopt`. Think of `CoinOS` as a *metasolver*. As an example (we assume a Windows operating system and use the .exe extension), consider:

```
gams.exe eastborne.gms MIP=CoinOS
```

The solver name `CoinOS` is not case sensitive and

```
gams.exe eastborne.gms MIP=coinos
```

will also work. In addition, if

```
Option MIP = CoinOS ;
```

is present in the GAMS file, then writing `MIP=CoinOS` on the command line is unnecessary. Since `Option MIP = CoinOS;` is present in the GAMS model file `eastborne.gms`, we will not specify it explicitly on the command line in the ensuing discussion. To summarize,

```
gams.exe eastborne.gms
```

is equivalent to the two versions of the command given previously. Executing any of the commands will result in the model being solved on the local machine using the COIN-OR solver `Cbc`, the default solver for mixed-integer linear models (MIP).

It is possible to control which solver is selected by `CoinOS`. This is done by providing an *options file* to GAMS. Since the solver is named `CoinOS`, the options file should be named `CoinOS.opt` (the file name is not case sensitive) and the command line call is

```
gams.exe eastborne.gms optfile 1
```

Calling multiple GAMS options files uses the convention

```
optfile=1 corresponds to CoinOS.opt
optfile=2 corresponds to CoinOS.op2
...
optfile=99 corresponds to CoinOS.o99
```

We now explain the valid options that can go into a GAMS option file when using the `CoinOS` solver. They are:

`solver (string)`: Specifies the solver that is used to solve an instance. Valid values are `clp`, `cbc`, `glpk`, `ipopt`, and `bonmin`. If a solver name is specified that is not recognized, the default solver for the problem type is used. The value for the solver option is case insensitive. For example, if the file `CoinOS.opt` contains a single line

```
solver glpk
```

then executing

```
gams.exe eastborne.gms optfile 1
```

will result in using `Glpk` to solve the problem.

`writeosil (string)`: If this option is used, GAMS will write the optimization instance to file `(string)` in OSiL format.

`writeosrl (string)`: If this option is used, GAMS will write the result of the optimization to file `(string)` in OSrL format.

The options just described are options for the GAMS modeling language. It is also possible to pass options directly to the COIN-OR solvers by using the `OS` interface. This is done by passing the name of an options file that conforms to the OSoL standard. The option

`readosol (string)` specifies the name of an OS option file in OSoL format that is given to the solver. Note: The file `CoinOS.opt` is an option file for GAMS but the GAMS option `readosol` in the GAMS options file is specifying the name of an OS options file.

The file `solveroptions.osol` is contained in the OS distribution in the `osolFiles` directory in the `data` directory. This file contains four solver options; two for `Cbc`, one for `Ipopt`, and one for `SYMPHONY` (which is available for remote server calls, but not locally). You can have any number of options. Note the format for specifying an option:

```
<solverOption name="maxN" solver="cbc" value="5" />
```

The attribute `name` specifies that the option name is `maxN` which is the maximum number of nodes allowed in the branch-and-bound tree, the `solver` attribute specifies the name of the solver to which the option should be applied, and the `value` attribute specifies the value of the option.

As a second example, consider the specification

```
<solverOption name="max_iter" solver="ipopt" type="integer" value="2000"/>
```

In this example we are specifying an iteration limit for `Ipopt`. Note the additional attribute `type` that has value `integer`. The Ipopt solver requires specifying the data type (string, integer, or numeric) for its options. For a list of options that solvers take, see the file

```
docs/solvers/coin.pdf
```

inside the GAMS directory. An up-to-date online version of this list is available at `http://www.coin-or.org/GAMSlinks/gamscoin.pdf`.

9.2.2 Using GAMS to Invoke a Remote OS Solver Service

We now describe how to call a remote OS solver service using the GAMS `CoinOS`. Before proceeding, it is important to emphasize that when calling a remote OS solver service, the remote service may be a different implementation of OS than the GAMS implementation in `CoinOS`. For example, the remote implementation may also provide access to solvers such as `SYMPHONY`, `Couenne`, and `DyLP`. There are several reason why you might wish to use a remote OS solver service.

- Have access to a faster machine.
- Be able to submit jobs to run in asynchronous mode – submit your job, turn off your laptop, and check later to see if the job ran.
- Call several additional solvers (`SYMPHONY`, `Couenne` and `DyLP`).

In order to use the COIN-OR solver service it is necessary to specify the service URL. This is done using the `service` option.

`service (string)`: Specifes the URL of the COIN-OR solver service

Use the following value for this option.

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
```

Default solver values are present, depending on the problem for characteristics. For more details, consult Table 4 (p.37). In order to control the solver used, it is necessary to specify the name of the solver inside the XML tag `<solverToInvoke>`. The example `solveroptions.osol` file contains the XML tag

```
<solverToInvoke>symphony</solverToInvoke>
```

If, for example, the `CoinOS.opt` file is

```
solver ipopt
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
readosol  solveroptions.osol
writeosrl temp.osrl
```

then `Ipopt` is ignored as a solver option and the remote server uses the `SYMPHONY` solver. Valid values for the remote solver service specified in the `<solverToInvoke>` tag are `clp`, `cbc`, `dylp`, `glpk`, `ipopt`, `bonmin`, `couenne`, `symphony`, and `vol`. If the problem is solved using a remote solver service the value specified by the GAMS `solver` option is irrelevant and ignored.

The GAMS `CoinOS` solver behaves differently from other implementiations of OS in the following way. Although it is possible to put the address of the remote server in the OS options file, it is not read by the GAMS `CoinOS` solver. The only way to specify a remote solver is through the GAMS `service` option.

By default, the call to the server is a *synchronous* call. The GAMS process will wait for the result and then display the result. This may not be desirable when solving large optimization models. The user may wish to submit a job, turn off his or her computer, and then check at a later date to see if the job is finished. In order to use the remote solver service in this fashion, i.e., *asynchronously*, it is necessary to use the `service_method` option.

`service_method (string)` specifies the method to execute on a server. Valid values for this option are `solve`, `getJobID`, `send`, `knock`, `retrieve`, and `kill`. We explain how to use each of these.

The default value of `service_method` is `solve`. A `solve` invokes the remote service in synchronous mode. When using the `solve` method you can optionally specify a set of solver options in an OSoL file by using the `readosol` option. The remaining values for the `service_method` option are used for an asynchronous call. We illustrate them in the order in which they would most logically be executed.

`service_method getJobID`: When working in asynchronous mode, the server needs to uniquely identify each job. The `getJobID` service method will result in the server returning a unique job id. For example if the following `CoinOS.opt` file is used

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method getJobID
```

with the command

```
gams.exe eastborne.gms optfile=1
```

the user will see a rather long job id returned to the screen as output. Assume that the job id returned is `coinor12345xyz`. This job id is used to submit a job to the server with the `send` method. Any job id can be sent to the server as long as it has not been used before.

`service_method send`: When working in asynchronous mode, use the `send` service method to submit a job. When using the `send` service method option an option is required and the options file must specify a job id that has not been used before. Assume that in the `CoinOS.opt` we specify the options:

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method send
readosol sendWithJobID.osol
```

The `sendWithJobID.osol` options file is identical to the `solveroptions.osol` options file except that it has an additional XML tag:

```
<jobID>coinor12345xyz</jobID>
```

We then execute

```
gams.exe eastborne.gms optfile=1
```

If all goes well, the response to the above command should be: "Problem instance successfully sent to OS service". At this point the server will schedule the job and work on it. It is possible to turn off the user computer at this point. At some point the user will want to know if the job is finished. This is accomplished using the `knock` service method.

`service_method knock`: When working in asynchronous mode, this is used to check the status of a job. Consider the following `CoinOS.opt` file:

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method knock
readosol sendWithJobID.osol
readospl knock.ospl
writeospl knockResult.ospl
```

The `knock` service method requires two inputs. The first input is the name of an options file, in this case `sendWithJobID.osol`, specified through the `readosol` option. In addition, a file in OSpL format is required. You can use the `knock.opsl` file provided in the binary distribution. This file name is specified using the `readospl` option. If no job id is specified in the OSoL file then the status of all jobs on the server will be returned in the file specified by the `writeospl` option. If a job id is specified in the OSoL file, then only information on the specified job id is returned in the file specified by the `writeospl` option. In this case the file name is `knockResult.ospl`. We then execute

```
gams.exe eastborne.gms optfile=1
```

The file `knockResult.ospl` will contain the information

```
<job jobID="coinor12345xyz">
<state>finished</state>
<serviceURI>http://192.168.0.219:8443/os/OSSolverService.jws</serviceURI>
<submitTime>2009-11-10T02:13:11.245-06:00</submitTime>
<startTime>2009-11-10T02:13:11.245-06:00</startTime>
<endTime>2009-11-10T02:13:12.605-06:00</endTime>
<duration>1.36</duration>
 </job>
```

Note that the job is complete as indicated in the `<state>` tag. It is now time to actually retrieve the job solution. This is done with the `retrieve` method.

`service_method retrieve`: When working in asynchronous mode, this method is used to retrieve the job solution. It is necessary when using `retrieve` to specify an options file and in that options file specify a job id. Consider the following `CoinOS.opt` file:

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method retrieve
readosol sendWithJobID.osol
writeosrl answer.osrl
```

When we then execute

```
gams.exe eastborne.gms optfile=1
```

the result is written to the file `answer.osrl`.

Finally there is a `kill` service method which is used to kill a job that was submitted by mistake or is running too long on the server.

`service_method kill:` When working in asynchronous mode, this method is used to terminate a job. You should specify an OSoL file containing the JobID by using the `readosol` option.

9.2.3 GAMS Summary:

1. In order to use OS with GAMS you can either specify `CoinOS` as an option to GAMS at the command line,

   ```
   gams eastborne.gms MIP=CoinOS
   ```

   or you can place the statement `Option ProblemType = CoinOS;` somewhere in the model *before* the `Solve` statement in the GAMS file.

2. If no options are given, then the model will be solved locally using the default solver (see Table 4 on p.37).

3. In order to control behavior (for example, whether a local or remote solver is used) an options file, `CoinOS.opt`, must be used as follows

   ```
   gams.exe  eastborne.gms optfile=1
   ```

4. The `CoinOS.opt` file is used to specify *eight potential options*:

- `service (string)`: using the COIN-OR solver server; this is done by giving the option

  ```
  service  http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
  ```

- `readosol (string)`: whether or not to send the solver an options file; this is done by giving the option

  ```
  readosol  solveroptions.osol
  ```

- `solver (string)`: if a local solve is being done, a specific solver is specified by the option

  ```
  solver solver_name
  ```

  Valid values are `clp`, `cbc`, `glpk`, `ipopt` and `bonmin`. When the COIN-OR solver service is being used, the only way to specify the solver to use is through the `<solverToInvoke>` tag in an OSoL file. In this case the valid values for the solver are `clp`, `cbc`, `dylp`, `glpk`, `ipopt`, `bonmin`, `couenne`, `symphony` and `vol`.

- `writeosrl (string)`: the solution result can be put into an OSrL file by specifying the option

  ```
  writeosrl  osrl_file_name
  ```

- `writeosil (string)`: the optimization instance can be put into an OSiL file by specifying the option

  ```
  writeosil  osil_file_name
  ```

- `writeospl (string)`: Specifies the name of an OSpL file in which the answer from the `knock` or `kill` method is written, e.g.,

  ```
  writeospl  write_ospl_file_name
  ```

- `readospl (string)`: Specifies the name of an OSpL file that the `knock` method sends to the server

  ```
  readospl  read_ospl_file_name
  ```

- `service_method (string)`: Specifies the method to execute on a server. Valid values for this option are `solve`, `getJobID`, `send`, `knock`, `retrieve`, and `kill`.

5. If an OS options file is passed to the GAMS `CoinOS` solver using the GAMS `CoinOS` option `readosol`, then GAMS does not interpret or act on any options in this file. The options in the OS options file are passed directly to either: i) the default local solver, ii) the local solver specified by the GAMS `CoinOS` option `solver`, or iii) to the remote OS solver service if one is specified by the GAMS `CoinOS` option `service`.

9.3 MATLAB: Using MATLAB to Build and Run OSiL Model Instances

MATLAB has powerful matrix generation and manipulation routines. This section is for users who wish to use MATLAB to generate the matrix coefficients for linear or quadratic programs and use the OS library to call a solver and get the result back. Using MATLAB with OS requires the user compile a file `OSMatlabSolverMex.cpp` into a MATLAB executable file (these files will have a `.mex` extension) after compilation. This executable file is linked to the OS library and works through the MATLAB API to communicate with the OS library.

The OS MATLAB application differs from the other applications in the `OS/applications` folder in that makefiles are not used. The file

`OS/applications/matlab/OSMatlabSolverMex.cpp`

must be compiled inside the MATLAB command window. Building the OS MATLAB application requires the following steps.

Step 1: The MATLAB installation contains a file `mexopts.sh` (UNIX) or `mexopts.bat` (Windows) that must be edited. This file typically resides in the `bin` directory of the MATLAB application. This file contains compile and link options that must be properly set. Appropriate paths to header files and libraries must be set. This discussion is based on the assumption that the user has either done a `make install` for the OS project or has downloaded a binary archive of the OS project. In either case there will be an `include` directory with the necessary header files and a `lib` directory with the necessary libraries for linking.

First edit the `CXXFLAGS` option to point to the header files in the `cppad` directory and the `include` directory in the project root. For example, it should look like:

```
CXXFLAGS='-fno-common -no-cpp-precomp -fexceptions
    -I/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OS/
    -I/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OS/include'
```

Next edit the `CXXLIBS` flag so that the OS and supporting libraries are included. For example, it should look like the following on a MacIntosh:

```
CXXLIBS="$MLIBS -lstdc++ -L/Users/kmartin/coin/os-trunk/vpath/lib
-lOS -lbonmin -lIpopt -lOsiCbc -lOsiClp -lOsiSym -lOsiVol
-lOsiDylp -lCbc -lCgl -lOsi -lClp  -lSym -lVol -lDylp
-lCoinUtils -lCbcSolver  -lcoinmumps -ldl -lpthread
/usr/local/lib/libgfortran.dylib -lgcc_s.10.5 -lgcc_ext.10.5 -lSystem -lm
```

Important: It has been the authors' experience that setting the necessary MATLAB compiler and linker options to build the `mex` can be tricky. We include in

`OS/applications/matlab/macOSXscript.txt`

the exact options that work on a 64 bit Mac with MATLAB release R2009b.

Step 2: Build the MATLAB executable file. Start MATLAB and in the MATLAB command window connect to the directory `OS/examples/matlab` which contains the file `OSmatlabSolver.cpp`. Execute the command:

```
mex -v OSMatlabSolver.cpp
```

On a 64 bit machine the command should be

```
mex -v -largeArrayDims OSMatlabSolver.cpp
```

On an Intel MAC OS X 64 bit chip the resulting executable will be named `OSmatlabSolver.mexmaci64`. On the Windows system the file is named `OSmatlabSolver.mexw32`.

Step 3: Set the MATLAB path to include the directory `OS/applications/matlab` (or more generally, the directory with the `mex` executable).

Step 4: In the MATLAB command window, connect to the directory `OS/data/matlabFiles`. Run either of the MATLAB files `markowitz.m` or `parincLinear.m`. The result should be displayed in the MATLAB browser window.

To use the `OSmatlabSolver` it is necessary to put the coefficients from a linear, integer, or quadratic problem into MATLAB arrays. We illustrate for the linear program:

$$\begin{aligned}
\text{Minimize} \quad & 10x_1 + 9x_2 && (5)\\
\text{Subject to} \quad & .7x_1 + x_2 \leq 630 && (6)\\
& .5x_1 + (5/6)x_2 \leq 600 && (7)\\
& x_1 + (2/3)x_2 \leq 708 && (8)\\
& .1x_1 + .25x_2 \leq 135 && (9)\\
& x_1, x_2 \geq 0 && (10)
\end{aligned}$$

The MATLAB representation of this problem in MATLAB arrays is

```
% the number of constraints
numCon = 4;
% the number of variables
numVar = 2;
% variable types
VarType='CC';
% constraint types
A = [.7  1; .5  5/6; 1   2/3  ; .1   .25];
BU = [630 600  708  135];
BL = [];
OBJ = [10  9];
VL = [-inf -inf];
VU = [];
ObjType = 1;
% leave Q empty if there are no quadratic terms
Q = [];
prob_name = 'ParInc Example'
password = '';
%
```

```
%
%the solver
solverName = 'ipopt';
%the remote service address
%if left empty we solve locally -- must solve locally for now
serviceAddress='';
% now solve
callMatlabSolver( numVar, numCon, A, BL, BU, OBJ, VL, VU, ObjType, ...
    VarType, Q, prob_name, password, solverName, serviceAddress)
```

This example m-file is in the `data` directory and is file `parincLinear.m`. Note that in addition to the problem formulation we can specify which solver to use through the `solverName` variable. If solution with a remote solver is desired this can be specified with the `serviceAddress` variable. If the `serviceAddress` is left empty, i.e.,

```
serviceAddress='';
```

then a local solver is used. In this case it is crucial that the appropriate solver is linked in with the `matlabSolver` executable using the `CXXLIBS` option.

The data directory also contains the m-file `template.m` which contains extensive comments about how to formulate the problems in MATLAB. The user can edit `template.m` as necessary and create a new instance.

A second example which is a quadratic problem is given in Section 9.3. The appropriate MATLAB m-file is `markowitz.m` in the data/matlabFiles directory. The problem consists in investing in a number of stocks. The expected returns and risks (covariances) of the stocks are known. Assume that the decision variables x_i represent the fraction of wealth invested in stock i and that no stock can have more than 75% of the total wealth. The problem then is to minimize the total risk subject to a budget constraint and a lower bound on the expected portfolio return.

Assume that there are three stocks (variables) and two constraints (not counting the upper limit of .75 on the investment variables).

```
% the number of constraints
numCon = 2;
% the number of variables
numVar = 3;
```

All the variables are continuous:

```
VarType='CCC';
```

Next define the constraint upper and lower bounds. There are two constraints, an equality constraint (an $=$) and a lower bound on portfolio return of .15 (a $\geq$). These two constraints are expressed as

```
BL = [1   .15];
BU = [1  inf];
```

The variables are nonnegative and have upper limits of .75 (no stock can comprise more than 75% of the portfolio). This is written as

```
VL = [];
VU = [.75 .75 .75];
```

There are no nonzero linear coefficients in the objective function, but the objective function vector must always be defined and the number of components of this vector is the number of variables.

```
OBJ = [0 0 0 ]
```

Now the linear constraints. In the model the two linear constraints are

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ 0.3221x_1 + 0.0963x_2 + 0.1187x_3 &\geq .15 \end{aligned}$$

These are expressed as

```
A = [ 1 1 1  ;
 0.3221   0.0963   0.1187 ];
```

Now for the quadratic terms. The only quadratic terms are in the objective function. The objective function is

$$\begin{aligned} \min 0.425349694x_1^2 + 0.445784443x_2^2 + 0.231430983x_3^2 + 2 \times 0.185218694x_1x_2 \\ +2 \times 0.139312545x_1x_3 + 2 \times 0.13881692x_2x_3 \end{aligned}$$

To represent quadratic terms MATLAB uses an array, here denoted Q, which has four rows, and a column for each quadratic term. In this example there are six quadratic terms. The first row of Q is the row index where the terms appear. By convention, the objective function has index -1, and constraints are counted starting at 0. The first row of Q is

```
-1 -1 -1 -1 -1 -1
```

The second row of Q is the index of the first variable in the quadratic term. We use zero based counting. Variable x_1 has index 0, variable x_2 has index 1, and variable x_3 has index 2. Therefore, the second row of Q is

```
0 1 2 0 0 1
```

The third row of Q is the index of the second variable in the quadratic term. Therefore, the third row of Q is

```
0 1 2 1 2 2
```

Note that terms such as x_1^2 are treated as $x_1 * x_1$ and that mixed terms such as x_2x_3 could be given in either order.

The last (fourth) row is the coefficient. Therefore, the fourth row reads

```
.425349654   .445784443   .231430983    .370437388   .27862509    .27763384
```

The full array is

```
Q = [ -1 -1 -1 -1 -1 -1;
      0 1 2 0 0 1 ;
      0 1 2 1 2 2;
      .425349654  .445784443  .231430983   .370437388  .27862509   .27763384
    ];
```

Finally, name the problem, specify the solver (in this case `ipopt`), the service address (and password if required by the service), and call the solver.

```
% replace Template with the name of your  problem
prob_name = 'Markowitz Example from Anderson, Sweeney, Williams, and Martin';
password = '';
%
%the solver
solverName = 'ipopt';
%the remote service service address
%if left empty we solve locally -- must solve locally for now
serviceAddress='';
% now solve
OSCallMatlabSolver( numVar, numCon, A, BL, BU, OBJ, VL, VU, ObjType, VarType, ...
     Q, prob_name, password, solverName, serviceAddress)
```

10 The OS Library Components

10.1 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). There are two key classes in the `OSAgent` component of the OS library. The two classes are `OSSolverAgent` and `WSUtil`.

The `OSSolverAgent` class is used to contact a remote solver service. For example, assume that `sOSiL` is a string with a problem instance and `sOSoL` is a string with solver options. Then the following code will call a solver service and invoke the `solve` method.

```
OSSolverAgent *osagent;
string serviceLocation = http://kipp.chicagobooth.edu/os/OSSolverService.jws
osagent = new OSSolverAgent(  serviceLocation );
string sOSrL = osagent->solve(sOSiL, sOSoL);
```

Other methods in the `OSSolverAgent` class are `send`, `retrieve`, `getJobID`, `knock`, and `kill`. The use of these methods is described in Section 7.4.

The methods in the `OSSolverAgent` class call methods in the `WSUtil` class that perform such tasks as creating and parsing SOAP messages and making low level socket calls to the server running the solver service. The average user will not use methods in the `WSUtil` class, but they are available to anyone wanting to make socket calls or create SOAP messages.

There is also a method, `OSFileUpload`, in the OSAgentClass that is used to upload files from the hard drive of a client to the server. It is very fast and does not involve SOAP or Web Services. The `OSFileUpload` method is illustrated and described in the example code `OSFileUpload.cpp` described in Section 14.

10.2 OSCommonInterfaces

The classes in the OSCommonInterfaces component of the OS library are used to read and write files and strings in the OSiL and OSrL protocols. See Section 6 for more detail on OSiL, OSrL, and other OS protocols. For a complete listing of all of the files in `OSCommonInterfaces` see the Doxygen documentation we deposited at `http://www.doxygen.org`. Users who have Doxygen installed on their system can also create their own version of the documentation (see Section 4.7). Below we highlight some key classes.

10.2.1 The OSInstance Class

The OSInstance class is the in-memory representation of an optimization instance and is a key class for users of the OS project. This class has an API defined by a collection of `get()` methods for extracting various components (such as bounds and coefficients) from a problem instance, a collection of `set()` methods for modifying or generating an optimization instance, and a collection of `calculate()` methods for function, gradient, and Hessian evaluations. See Section 11. We now describe how to create an `OSInstance` object and the close relationship between the OSiL schema and the `OSInstance` class.

10.2.2 Creating an `OSInstance` Object

The OSCommonInterfaces component contains an `OSiLReader` class for reading an instance in an OSiL string and creating an in-memory `OSInstance` object. Assume that `sOSiL` is a string that will hold the instance in OSiL format. Creating an `OSInstance` object is illustrated in Figure 13.

```
OSiLReader *osilreader = NULL;
OSInstance *osinstance = NULL;
osilreader = new OSiLReader();
osinstance = osilreader->readOSiL( sOSiL);
```

Figure 13: Creating an `OSInstance` Object

10.2.3 Mapping Rules

The `OSInstance` class has two members, `instanceHeader` and `instanceData`. These correspond to the XML elements `<instanceHeader>` and `<instanceData>`. They are of type `InstanceHeader` and `InstanceData`, respectively, which in turn correspond to the OSiL schema's complexTypes `InstanceHeader` and `InstanceData`, and in themselves are C++ classes.

Moving down one level, Figure 15 shows that the `InstanceData` class has in turn the members `variables`, `objectives`, `constraints`, `linearConstraintCoefficients`, `quadraticCoefficients`, and `nonlinearExpressions`, corresponding to the respective elements in the OSiL file that have the same name. Each of these are instances of associated classes which correspond to complexTypes in the OSiL schema.

Figure 16 uses the `Variables` class to provide a closer look at the correspondence between schema and class. On the right, the `Variables` class contains the data member `numberOfVariables` and a sequence of `var` objects of class `Variable`. The `Variable` class has `lb` (double), `ub` (double),

```
class OSInstance{
public:
     OSInstance();
     InstanceHeader *instanceHeader;
     InstanceData *instanceData;
}; //class OSInstance
```

Figure 14: The `OSInstance` class

```
class InstanceData{
public:
     InstanceData();
     Variables *variables;
     Objectives *objectives;
     Constraints *constraints;
     LinearConstraintCoefficients *linearConstraintCoefficients;
     QuadraticCoefficients *quadraticCoefficients;
     NonlinearExpressions *nonlinearExpressions;
}; // class InstanceData
```

Figure 15: The `InstanceData` class

`name` (string), and `type` (char) data members. On the left the corresponding XML complexTypes are shown, with arrows indicating the correspondences. The following rules describe the mapping between the OSiL schema and the `OSInstance` class.

- ▷ Each complexType in an OSiL schema corresponds to a class in `OSInstance.` Thus the OSiL schema's complexType `Variables` corresponds to `OSInstance`'s class `Variables`. Elements in an actual XML file then correspond to objects in `OSInstance`; for example, the `<variables>` element that is of type `Variables` in an OSiL file corresponds to a `variables` object in class `Variables` of `OSInstance`.

- ▷ An attribute or element used in the definition of a `complexType` is a member of the corresponding `OSInstance` class, and the type of the attribute or element matches the type of the member. In Figure 16, for example, `lb` is an attribute of the OSiL `complexType` named `Variable`, and `lb` is a member of the `OSInstance` class `Variable`; both have type `double`. Similarly, `<var>` is an element in the definition of the OSiL `complexType` named `Variables`, and `var` is a member of the `OSInstance` class `Variables`; the `<var>` element has type `Variable` and the `var` member is a `Variable` object.

- ▷ A schema sequence corresponds to an array. For example, in Figure 16 the complexType `Variables` has a sequence of `<var>` elements that are of type `Variable`, and the corresponding `Variables` class has a member that is an array of type `Variable`.

General nonlinear terms are stored in the data structure as `OSExpressionTree` objects, which are the subject of the next section.

The `OSInstance` class has a collection of `get()`, `set()`, and `calculate()` methods that act as an API for the optimization instance and are described in Section 11.

10.2.4 The OSExpressionTree OSnLNode Classes

The `OSExpressionTree` class provides the in-memory representation of the nonlinear terms. Our design goal is to allow for efficient parsing of OSiL instances, while providing an API that meets the needs of diverse solvers. Conceptually, any nonlinear expression in the objective or constraints is represented by a tree. The expression tree for the nonlinear part of the objective function (1), for example, has the form illustrated in Figure 17. The choice of a data structure to store such a tree — along with the associated methods of an API — is a key aspect in the design of the `OSInstance` class.

A base abstract class `OSnLNode` is defined and all of an OSiL file's operator and operand elements used in defining a nonlinear expression are extensions of the base element type `OSnLNode`. There

Schema complexType **In-memory class**

```
<xs:complexType name="Variables">  <-------------------------------------------->  class Variables{
                                                                                   public:
  <xs:sequence>                                                                      Variables();
    <xs:element name="var" type="Variable" maxOccurs="unbounded"/>  <----------->    Variable *var;
  </xs:sequence>
  <xs:attribute name="numberOfVariables" type="xs:positiveInteger"
                use="required"/>  <-------------------------------------------->    int numberOfVariables;
</xs:complexType>                                                                  }; // class Variables


<xs:complexType name="Variable">  <-------------------------------------------->  class Variable{
                                                                                   public:
                                                                                     Variable();
  <xs:attribute name="name" type="xs:string" use="optional"/>  <---------------->    string name;
  <xs:attribute name="type" use="optional" default="C">  <---------------------->    char type;
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="C"/>
        <xs:enumeration value="B"/>
        <xs:enumeration value="I"/>
        <xs:enumeration value="S"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="lb" type="xs:double" use="optional" default="0"/>  <------>    double lb;
  <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>  <---->    double ub;
</xs:complexType>                                                                  }; // class Variable
```

OSiL elements **In-memory objects**

```
<variables numberOfVariables="2">
   <var lb="0" name="x0" type="C"/>
   <var lb="0" name="x1" type="C"/>
</variables>
```

```
OSInstance osinstance;
osinstance.instanceData.variables.numberOfVariables=2;
osinstance.instanceData.variables.var=new Var[2];
osinstance.instanceData.variables.var[0].lb=0;
osinstance.instanceData.variables.var[0].name=x0;
osinstance.instanceData.variables.var[0].type=C;
osinstance.instanceData.variables.var[1].lb=0;
osinstance.instanceData.variables.var[1].name=x1;
osinstance.instanceData.variables.var[1].type=C;
```

Figure 16: The `<variables>` element as an `OSInstance` object

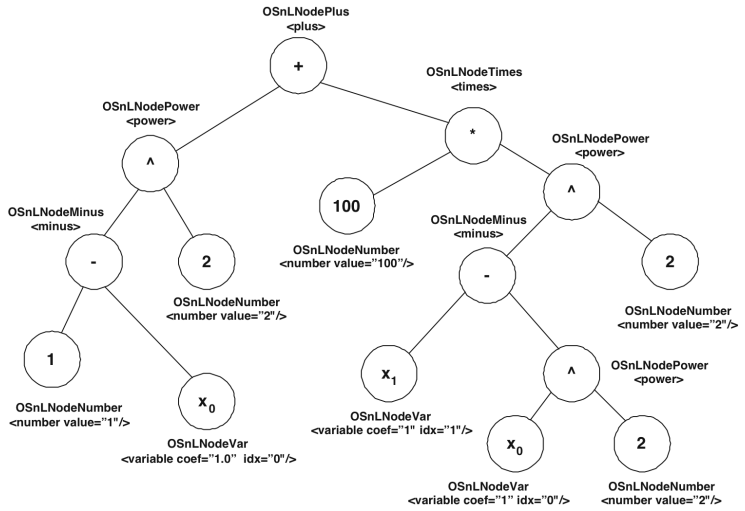

Figure 17: Conceptual expression tree for the nonlinear part of the objective (1).

is an element type `OSnLNodePlus`, for example, that extends `OSnLNode`; then in an OSiL instance file, there are `<plus>` elements that are of type `OSnLNodePlus`. Each `OSExpressionTree` object contains a pointer to an `OSnLNode` object that is the root of the corresponding expression tree. To every element that extends the `OSnLNode` type in an OSiL instance file, there corresponds a class that derives from the `OSnLNode` class in an `OSInstance` data structure. Thus we can construct an expression tree of homogenous nodes, and methods that operate on the expression tree to calculate function values, derivatives, postfix notation, and the like do not require switches or complicated logic.

```
double OSnLNodePlus::calculateFunction(double *x){
   m_dFunctionValue =
      m_mChildren[0]->calculateFunction(x) +
      m_mChildren[1]->calculateFunction(x);
   return m_dFunctionValue;
} //calculateFunction
```

Figure 18: The function calculation method for the `plus` node class with polymorphism

The `OSInstance` class has a variety of `calculate()` methods, based on two pure virtual functions in the `OSInstance` class. The first of these, `calculateFunction()`, takes an array of `double` values corresponding to decision variables, and evaluates the expression tree for those values. Every class that extends `OSnLNode` must implement this method. As an example, the `calculateFunction` method for the `OSnLNodePlus` class is shown in Figure 18. Because the OSiL instance file must be validated against its schema, and in the schema each `<OSnLNodePlus>` element is specified to have exactly two child elements, this `calculateFunction` method can assume that there are exactly two children of the node that it is operating on. The use of polymorphism and recursion makes adding new operator elements easy; it is simply a matter of adding a new class and implementing the `calculateFunction()` method for it.

Although in the OSnL schema, there are 200+ nonlinear operators, only the following `OSnLNode` classes are currently supported in our implementation.

- OSnLNodeVariable
- OSnLNodeTimes
- OSnLNodePlus
- OSnLNodeSum
- OSnLNodeMinus
- OSnLNodeNegate
- OSnLNodeDivide
- OSnLNodePower
- OSnLNodeProduct
- OSnLNodeLn
- OSnLNodeSqrt
- OSnLNodeSquare
- OSnLNodeSin
- OSnLNodeCos
- OSnLNodeExp
- OSnLNodeIf
- OSnLNodeAbs
- OSnLNodeMax
- OSnLNodeMin
- OSnLNodeE
- OSnLNodePI
- OSnLNodeAllDiff

10.2.5 The OSOption Class

The `OSOption` class is the in-memory representation of the options associated with a particular optimization task. It is another key class for users of the OS project. This class has an API defined by a collection of `get()` methods for extracting various components (such as initial values for decision variables, solver options, job parameters, etc.), and a collection of `set()` methods for modifying or generating an option instance. The relationship between in-memory classes and objects on one hand and complexTypes and elements of the OSoL schema follow the same mapping rules laid out in Section 10.2.3.

10.2.6 The OSResult Class

Similarly the `OSResult` class is the in-memory representation of the results returned by the solver and other information associated with a particular optimization task. This class has an API defined by a collection of `set()` methods that allow a solver to create a result instance and a collection of `get()` methods for extracting various components (such as optimal values for decision variables, optimal objective function value, optimal dual variables, etc.). The relationship between in-memory classes and objects on one hand and complexTypes and elements of the OSoL schema follow the same mapping rules laid out in Section 10.2.3.

10.3 OSModelInterfaces

This part of the OS library is designed to help integrate the OS standards with other standards and modeling systems.

10.3.1 Converting MPS Files

The MPS standard is still a popular format for representing linear and integer programming problems. In `OSModelInterfaces`, there is a class `OSmps2osil` that can be used to convert files in MPS format into the OSiL standard. It is used as follows.

```
OSmps2osil *mps2osil = NULL;
DefaultSolver *solver  = NULL;
solver = new CoinSolver();
solver->sSolverName = "cbc";
mps2osil = new OSmps2osil(  mpsFileName);
mps2osil->createOSInstance() ;
solver->osinstance = mps2osil->osinstance;
solver->solve();
```

The `OSmps2osil` class constructor takes a string which should be the file name of the instance in MPS format. The constructor then uses the `CoinUtils` library to read and parse the MPS file. The class method `createOSInstance` then builds an in-memory `osinstance` object that can be used by a solver.

10.3.2 Converting AMPL nl Files

AMPL is a popular modeling language that saves model instances in the AMPL nl format. The `OSModelInterfaces` library provides a class, `OSnl2osil`, for reading an nl file and creating a corresponding in-memory `osinstance` object. It is used as follows.

```
OSnl2osil *nl2osil = NULL;
DefaultSolver *solver  = NULL;
solver = new LindoSolver();
nl2osil = new OSnl2osil( nlFileName);
nl2osil->createOSInstance() ;
solver->osinstance = nl2osil->osinstance;
solver->solve();
```

The OSnl2osil class works much like the OSmps2osil class. The OSnl2osil class constructor takes a string which should be the file name of the instance in nl format. The constructor then uses the AMPL ASL library routines to read and parse the nl file. The class method createOSInstance then builds an in-memory osinstance object that can be used by a solver.

In Section 9.1 we describe the OSAmplClient executable that acts as a "solver" for AMPL. The OSAmplClient uses the OSnl2osil class to convert the instance in nl format to OSiL format before calling a solver either locally or remotely.

10.4 OSParsers

The OSParsers component of the OS library contains reentrant parsers that read OSiL, OSoL and OSrL strings and build, respectively, in-memory OSInstance, OSOption and OSResult objects.

The OSiL parser is invoked through an OSiLReader object as illustrated below. Assume osil is a string with the problem instance.

```
OSiLReader *osilreader = NULL;
OSInstance *osinstance = NULL;
osilreader = new OSiLReader();
osinstance = osilreader->readOSiL( osil);
```

The readOSiL method has a single argument which is a (pointer to a) string. The readOSiL method then calls an underlying method yygetOSInstance that parses the OSiL string. The major components of the OSiL schema recognized by the parser are

```
<instanceHeader>
<instanceData>
<variables>
<objectives>
<constraints>
<linearConstraintCoefficients>
<quadraticCoefficients>
<nonlinearExpressions>
```

There are other components in the OSiL schema, but they are not yet implemented. In most large-scale applications the <variables>, <objectives>, <constraints>, and <linearConstraintCoefficients> will comprise the bulk of the instance memory. Because of this, we have "hard-coded" the OSiL parser to read these specific elements very efficiently. The parsing of the <quadraticCoefficients> and <nonlinearExpressions> is done using code generated by flex and bison. The file OSParseosil.l is used by flex to generate OSParseosil.cpp and the file OSParseosil.y is used by bison to generate OSParseosil.tab.cpp. In OSParseosil.l we use the reentrant option and in OSParseosil.y we use the pure-parser option to generate reentrant parsers. The OSParseosil.y file contains both our "hard-coded" parser and the grammar rules for the <quadraticCoefficients> and <nonlinearExpressions> sections. We are currently using GNU bison version 3.2 and flex 2.5.33.

The typical OS user will have no need to edit either OSParseosil.l or OSParseosil.y and therefore will not have to worry about running either flex or bison to generate the parsers. The generated parser code from flex and bison is distributed with the project and works on all of the platforms listed in Table 1. If the user does edit either parseosil.l or parseosil.y then parseosil.cpp and parseosil.tab.cpp need to be regenerated with flex and bison. If these programs are present, in the OS directory execute

```
make  run_parsers
```

(This requires Unix or a unix-like environment (Cygwin, MinGW, MSYS, etc.) under Windows.)

The files `OSParseosrl.l` and `OSParseosrl.y` are used by `flex` and `bison` to generate the code `OSParseosrl.cpp` and `OSParseosrl.tab.cpp` for parsing strings in OSrL format. The comments made above about the OSiL parser apply to the OSrL parser. The OSrL parser, like the OSiL parser, is invoked using an `OSrL` reading object. This is illustrated below (`osrl` is a string in OSrL format).

```
OSrLReader *osrlreader = NULL;
osrlreader = new OSrLReader();
OSResult *osresult = NULL;
osresult = osrlreader->readOSrL( osrl);
```

The OSoL parser follows the same layout and rules. The files `OSParseosol.l` and `OSParseosol.y` are used by `flex` and `bison` to generate the code `OSParseosol.cpp` and `OSParseosol.tab.cpp` for parsing strings in OSoL format. The OSoL parser is invoked using an `OSoL` reading object. This is illustrated below (`osol` is a string in OSoL format).

```
OSoLReader *osolreader = NULL;
osolreader = new OSoLReader();
OSOption *osoption = NULL;
osoption = osolreader->readOSoL( osol);
```

There is also a lexer `OSParseosss.l` for tokenizing the command line for the OSSolverService executable described in Section 7.

10.5 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 the commercial solver LINDO. The OS library has been tested with the following solvers using the Osi Interface.

- Bonmin
- Cbc
- Clp
- Couenne
- Cplex
- DyLP
- Glpk
- Ipopt
- SYMPHONY

- Vol

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;
OSOption   *osoption;
```

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 create 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;
```

Commercial solvers like LINDO do not have a COIN-OR `Osi` interface, but it is possible to write wrappers so that they can be 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();
```

A similar call is used for `Ipopt`. In this case, the `IpoptSolver` class inherits from both the `DefaultSolver` class and the Ipopt `TNLP` class. See

small`https://projects.coin-or.org/Ipopt/browser/stable/3.5/Ipopt/doc/documentation.pdf?format=r`

for more information on the Ipopt solver C++ implementation and the `TNLP` class.

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 12.2.

10.6 OSUtils

The OSUtils component of the OS library contains utility codes. For example, the `FileUtil` class contains useful methods for reading files into `string` or `char*` and writing files from `string` and `char*`. The `OSDataStructures` class holds other classes for things such as sparse vectors, sparse Jacobians, and sparse Hessians. The `MathUtil` class contains a method for converting between sparse matrices in row and column major form.

11 The OSInstance API

The OSInstance API can be used to:

- get information about model parameters, or convert the `OSExpressionTree` into a prefix or postfix representation through a collection of `get()` methods,
- modify, or even create an instance from scratch, using a number of `set()` methods,
- provide information to solvers that require function evaluations, Jacobian and Hessian sparsity patters, function gradient evaluations, and Hessian evaluations.

11.1 Get Methods

The `get()` methods are used by other classes to access data in an existing `OSInstance` object or get an expression tree representation of an instance in postfix or prefix format. Assume `osinstance` is an object in the `OSInstance` class created as illustrated in Figure 13. Then, for example,

```
osinstance->getVariableNumber();
```

will return an integer which is the number of variables in the problem,

```
osinstance->getVariableTypes();
```

will return a `char` pointer to the variable types (`C` for continuous, `B` for binary, and `I` for general integer),

```
getVariableLowerBounds();
```

will return a `double` pointer to the lower bound on each variable. There are similar `get()` methods for the constraints. There are numerous `get()` methods for the data in the `<linearConstraintCoefficients>` element, the `<quadraticCoefficients>` element, and the `<nonlinearExpressions>` element.

When an `osinstance` object is created, it is stored as an expression tree in an `OSExpressionTree` object. However, some solver APIs (e.g., LINDO) may take the data in a different format such as postfix and prefix. There are methods to return the data in either postfix or prefix format.

First define a `vector` of pointers to `OSnLNode` objects.

```
std::vector<OSnLNode*> postfixVec;
```

then get the expression tree for the objective function (index = -1) as a postfix vector of nodes.

```
postfixVec = osinstance->getNonlinearExpressionTreeInPostfix( -1);
```

If, for example, the `osinstance` object was the in-memory representation of the instance illustrated in Section 16.1 and Figure 17 then the code

```
for (i = 0 ; i < n; i++){
    cout << postfixVec[i]->snodeName << endl;
}
```

will produce

```
number
variable
minus
number
power
number
variable
variable
number
power
minus
number
power
times
plus
```

This postfix traversal of the expression tree in Figure 17 lists all the nodes by recursively processing all subtrees, followed by the root node. The method `processNonlinearExpressions()` in the `LindoSolver` class in the `OSSolverInterfaces` library component illustrates the use of a postfix vector of `OSnLNode` objects to build a Lindo model instance.

11.2 Set Methods

The `set()` methods can be used to build an in-memory `OSInstance` object. A code example of how to do this is in Section 12.2.

11.3 Calculate Methods

The `calculate()` methods are described in Section 13.

11.4 Modifying an `OSInstance` Object

The OSInstance API is designed to be used to either build an in-memory `OSInstance` object or provide information about the in-memory object (e.g., the number of variables). This interface is not designed for problem modification. We plan on later providing an `OSModification` object for this task. However, by directly accessing an `OSInstance` object it is possible to modify parameters in the following classes:

- `Variables`
- `Objectives`
- `Constraints`
- `LinearConstraintCoefficients`

For example, to modify the first nonzero objective function coefficient of the first objective function to 10.7 the user would write,

```
osinstance->instanceData->objectives->obj[0]->coef[0]->value = 10.7;
```

If the user wanted to modify the actual number of nonzero coefficients as declared by

```
osinstance->instanceData->objectives->obj[0]->numberOfObjCoef;
```

then the only safe course of action would be to delete the current `OSInstance` object and build a new one with the modified coefficients. It is strongly recommend that no changes are made involving allocated memory – i.e., any kind of `numberOf***`. Modifying an objective function coefficient is illustrated in the OSModDemo example. See Section 12.4.

After modifying an `OSInstance` object, it is necessary to set certain boolean variables to true in order for these changes to get reflected in the OS solver interfaces.

- `Variables` – if any changes are made to a parameter in this class set

  ```
  osinstance->bVariablesModified = true;
  ```

- `Objectives` – if any changes are made to a parameter in this class set

  ```
  osinstance->bObjectivesModified = true;
  ```

- `Constraints` – if any changes are made to a parameter in this class set

  ```
  osinstance->bConstraintsModified = true;
  ```

- `LinearConstraintCoefficients` – if any changes are made to a parameter in this class set

  ```
  osinstance->bAMatrixModified = true;
  ```

At this point, if the user desires to modify an `OSInstance` object that contains nonlinear terms, the only safe strategy is to delete the object and build a new object that contains the modifications.

11.5 Printing a Model for Debugging

The OSiL representation for the test problem `rosenbrockmod.osil` is given in Appendix 16.1. Many users will not find the OSiL representation useful for model debugging purposes. For users who wish to see a model in a standard infix representation we provide a method `printModel()`. Assume that we have an `osinstance` object in the `OSInstance` class that represents the model of interest. The call

```
osinstance->printModel( -1)
```

will result in printing the (first) objective function indexed by -1. In order to print constraint k use

```
osinstance->printModel( k)
```

In order to print the entire model use

```
osinstance->printModel( )
```

Below we give the result of `osintance->printModel( )` for the problem `rosenbrockmod.osil`.

```
Objectives:
min 9*x_1 + (((1 - x_0) ^ 2) + (100*((x_1 - (x_0 ^ 2)) ^ 2)))

Constraints:
(((((10.5*x_0)*x_0) + ((11.7*x_1)*x_1)) + ((3*x_0)*x_1)) + x_0) <= 25
10 <= ((ln( (x_0*x_1)) + (7.5*x_0)) + (5.25*x_1))

Variables:
x_0 Type = C  Lower Bound =  0  Upper Bound =  1.7976931348623157e308
x_1 Type = C  Lower Bound =  0  Upper Bound =  1.7976931348623157e308
```

12 Code samples to illustrate the OS Project

These example executable files are not built by running `configure` and `make`. In order to build the examples in a unix environment the user must first run

```
make install
```

in the COIN-OS project root directory (the discussion in this section assumes that the project root directory is `COIN-OS`). Running `make install` will place all the header files required by the examples in the directory

```
COIN-OS/include
```

and all of the libraries required by the examples in the directory

```
COIN-OS/lib
```

The source code for the examples is in the directory `COIN-OS/OS/examples`. For instance, the `osModDemo` example is in the directory

```
COIN-OS/OS/examples/osModDemo
```

Next, the user should connect to the appropriate example directory and run `make`. If the user has done a VPATH build, the makefiles will be in each respective example directory under

```
vpath_root/OS/examples
```

otherwise, the makefiles will be in each respective example directory under

```
COIN-OS/OS/examples
```

The `Makefile` in each example directory is fairly simple and is designed to be easily modified by the user if necessary. The part of the Makefile to be adjusted, if necessary, is

```
##########################################################################
#    You can modify this example makefile to fit for your own program.   #
#    Usually, you only need to change the five CHANGEME entries below.   #
##########################################################################
```

```
# CHANGEME: This should be the name of your executable
EXE = OSModDemo
# CHANGEME: Here is the name of all object files corresponding to the source
#           code that you wrote in order to define the problem statement
OBJS =  OSModDemo.o
# CHANGEME: Additional libraries
ADDLIBS =
# CHANGEME: Additional flags for compilation (e.g., include flags)
ADDINCFLAGS =  -I${prefix}/include
# CHANGEME: SRCDIR is the path to the source code; VPATH is the path to
# the executable. It is assumed # that the lib directory is in prefix/lib
# and the header files are inprefix/include
SRCDIR = /Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OS/OS/examples/osModDemo
VPATH = /Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OS/OS/examples/osModDemo
prefix = /Users/kmartin/Documents/files/code/cpp/OScpp/vpath
```

Developers can use the Makefiles as a starting point for building applications that use the OS project libraries.

Users of Microsoft Visual Studio can obtain the executables by opening the solution file `OS.sln` in Visual Studio (or by double-clicking on the file in Windows Explorer). Once the file is opened, select the Configuration Manager from the Build menu and select the projects you desire to be built. Then select Build Solution from the Build menu (or press F7).

The executables are also part of the binary distribution described in Section 4.2.2.

12.1 Algorithmic Differentiation: Using the OS Algorithmic Differentiation Methods

In the `OS/examples/algorithmicDiff` folder is test code `OSAlgorithmicDiffTest.cpp`. This code illustrates the key methods in the `OSInstance` API that are used for algorithmic differentiation. These methods are described in Section 13.

12.2 Instance Generator: Using the OSInstance API to Generate Instances

This example is found in the `instanceGenerator` folder in the `examples` folder. This example illustrates how to build a complete in-memory model instance using the `OSInstance` API. See the code `OSInstanceGenerator.cpp` for the complete example. Here we provide a few highlights to illustrate the power of the API.

The first step is to create an `OSInstance` object.

```
OSInstance *osinstance;
osinstance = new OSInstance();
```

The instance has two variables, x_0 and x_1. Variable x_0 is a continuous variable with lower bound of -100 and upper bound of 100. Variable x_1 is a binary variable. First declare the instance to have two variables.

```
osinstance->setVariableNumber( 2);
```

Next, add each variable. There is an `addVariable` method with the signature

```
addVariable(int index, string name, double lowerBound, double upperBound, char type);
```

Then the calls for these two variables are

```
osinstance->addVariable(0, "x0", -100, 100, 'C');
osinstance->addVariable(1, "x1", 0, 1, 'B');
```

There is also a method `setVariables` for adding more than one variable simultaneously. The objective function(s) and constraints are added through similar calls.

Nonlinear terms are also easily added. The following code illustrates how to add a nonlinear term $x_0 * x_1$ in the `<nonlinearExpressions>` section of OSiL. This term is part of constraint 1 and is the second of six constraints contained in the instance.

```
osinstance->instanceData->nonlinearExpressions->numberOfNonlinearExpressions = 6;
osinstance->instanceData->nonlinearExpressions->nl = new Nl*[ 6 ];
osinstance->instanceData->nonlinearExpressions->nl[ 1] = new Nl();
osinstance->instanceData->nonlinearExpressions->nl[ 1]->idx = 1;
osinstance->instanceData->nonlinearExpressions->nl[ 1]->osExpressionTree =
new OSExpressionTree();
// the nonlinear expression is stored as a vector of nodes in postfix format
// create a variable nl node for x0
nlNodeVariablePoint = new OSnLNodeVariable();
nlNodeVariablePoint->idx=0;
nlNodeVec.push_back( nlNodeVariablePoint);
// create the nl node for x1
nlNodeVariablePoint = new OSnLNodeVariable();
nlNodeVariablePoint->idx=1;
nlNodeVec.push_back( nlNodeVariablePoint);
// create the nl node for *
nlNodePoint = new OSnLNodeTimes();
nlNodeVec.push_back( nlNodePoint);
// now the expression tree
osinstance->instanceData->nonlinearExpressions->nl[ 1]->osExpressionTree->m_treeRoot =
nlNodeVec[ 0]->createExpressionTreeFromPostfix( nlNodeVec);
```

12.3 branchCutPrice: Using Bcp

This example illustrates the use of the COIN-OR Bcp (Branch-cut-and-price) project. This project offers the user with the ability to have control over each node in the branch and process. This makes it possible to add user-defined cuts and/or user-defined variables. At each node in the tree, a call is made to the method `process_lp_result()`. In the example problem we illustrate 1) adding COIN-OR Cgl cuts, 2) a user-defined cut, and 3) a user-defined variable.

12.4 OSModificationDemo: Modifying an In-Memory `OSInstance` Object

The `osModificationDemo` folder holds the file `OSModificationDemo.cpp`. This is similar to the `instanceGenerator` example. In this case, a simple linear program is generated. However, this example also illustrates how to modify an in-memory OSInstance object. In particular, we illustrate how to modify an objective function coeffient. Note the dual occurrence of the following code

```
solver->osinstance->bObjectivesModified = true;
```

in the `OSModificationDemo.cpp` file (lines 177 and 187). This line is critical, since otherwise changes made to the OSInstance object will not be passed to the solver.

This example also illustrates calling a COIN-OR solver, in this case `Clp`.

Important: the ability to modify a problem instance is still extremely limited in this release. A better API for problem modification will come with a later release of OS.

12.5 OSSolverDemo: Building In-Memory Solver and Option Objects

The code in the example file `OSSolverDemo.cpp` in the folder `osSolverDemo` illustrates how to build solver interfaces and an in-memory `OSOption` object. In this example we illustrate building a solver interface and corresponding `OSOption` object for the solvers `Clp`, `Cbc`, `SYMPHONY`, `Ipopt`, `Bonmin`, and `Couenne`. Each solver class inherits from a virtual `OSDefaultSolver` class. Each solver class has the string data members

- `osil` -- this string conforms to the OSiL standard and holds the model instance.
- `osol` -- this string conforms to the OSoL standard and holds an instance with the solver options (if there are any); this string can be empty.
- `osrl` -- this string conforms to the OSrL standard and holds the solution instance; each solver interface produces an osrl string.

Corresponding to each string there is an in-memory object data member, namely

- `osinstance` -- an in-memory `OSInstance` object containing the model instance and get() and set() methods to access various parts of the model.
- `osoption` -- an in-memory `OSOption` object; solver options can be accessed or set using get() and set() methods.
- `osresult` -- an in-memory `OSResult` object; various parts of the model solution are accessible through get() and set() methods.

For each solver we detail five steps:

Step 1: Read a model instance from a file and create the corresponding `OSInstance` object. For four of the solvers we read a file with the model instance in OSiL format. For the Clp example we read an MPS file and convert to OSiL. For the Couenne example we read an AMPL nl file and convert to OSiL.

Step 2: Create an `OSOption` object and set options appropriate for the given solver. This is done by defining

```
OSOption* osoption = NULL;
osoption = new OSOption();
```

A key method in the `OSOption` interface is `setAnotherSolverOption()`. This method takes the following arguments in order.

`std::string name` – the option name;

`std::string value` – the value of the option;

`std::string solver` – the name of the solver to which the option applies;

`std::string category` – options may fall into categories. For example, consider the Couenne solver. This solver is also linked to the Ipopt and Bonmin solvers and it is possible to set options for these solvers through the Couenne API. In order to set an Ipopt option you would set the `solver` argument to `couenne` and set the `category` option to `ipopt`.

`std::string type` – many solvers require knowledge of the data type, so you can set the type to `double`, `integer`, `boolean` or `string`, depending on the solver requirements. Special types defined by the solver, such as the type `numeric` used by the Ipopt solver, can also be accommodated. It is the user's responsibility to verify the type expected by the solver.

`std::string description` – this argument is used to provide any detail or additional information about the option. An empty string (`""`) can be passed if such additional information is not needed.

For excellent documentation that details solver options for Bonmin, Cbc, and Ipopt we recommend

`http://www.coin-or.org/GAMSlinks/gamscoin.pdf`

Step 3: Create the solver object. In the OS project there is a *virtual* solver that is declared by

```
DefaultSolver *solver  = NULL;
```

The Cbc, Clp and SYMPHONY solvers as well as other solvers of linear and integer linear programs are all invoked by creating a `CoinSolver()`. For example, the following is used to invoke Cbc.

```
solver = new CoinSolver();
solver->sSolverName ="cbc";
```

Other solvers, particularly Ipopt, Bonmin and Couenne are implemented separately. So to declare, for example, an Ipopt solver, one should write

```
solver = new IpoptSolver();
```

The syntax is the same regardless of solver.

Step 4: Import the `OSOption` and `OSInstance` into the solver and solve the model. This process is identical regardless of which solver is used. The syntax is:

```
solver->osinstance = osinstance;
solver->osoption = osoption;
solver->solve();
```

Step 5: After optimizing the instance, each of the OS solver interfaces uses the underlying solver API to get the solution result and write the result to a string named `osrl` which is a string representing the solution instance in the `OSrL` XML standard. This string is accessed by

```
solver->osrl
```

In the example code `OSSolverDemo.cpp` we have written a method,

```
void getOSResult(std::string osrl)
```

that takes the `osrl` string and creates an `OSResult` object. We then illustrate several of the `OSResult` API methods

```
double getOptimalObjValue(int objIdx, int solIdx);
std::vector<IndexValuePair*>  getOptimalPrimalVariableValues(int solIdx);
```

to get and write out the optimal objective function value, and optimal primal values. See also Section 12.6.

We now highlight some of the features illustrated by each of the solver examples.

- **Clp** – In this example we read in a problem instance in MPS format. The class `OSmps2osil` has a method `mps2osil` that is used to convert the MPS instance contained in a file into an in-memory `OSInstance` object. This example also illustrates how to set options using the Osi interface. In particular we turn on intermediate output which is turned off by default in the Coin Solver Interface.

- **Cbc** – In this example we read a problem instance that is in OSiL format and create an in-memory `OSInstance` object. We then create an `OSOption` object. This is quite trivial. A plain-text XML file conforming to the OSiL schema is read into a string `osil` which is then converted into the in-memory `OSInstance` object by

```
OSiLReader *osilreader = NULL;
OSInstance *osinstance = NULL;
osilreader = new OSiLReader();
osinstance = osilreader->readOSiL( osil);
```

We set the linear programming algorithm to be the primal simplex method and then set the option on the pivot selection to be Dantzig rule. Finally, we set the print level to be 10.

- **SYMPHONY** – In this example we also read a problem instance that is in OSiL format and create an in-memory `OSInstance` object. We then create an `OSOption` object and illustrate setting the `verbosity` option.

- **Ipopt** – In this example we also read a problem instance that is in OSiL format. However, in this case we do not create an `OSInstance` object. We read the OSiL file into a string `osil`. We then feed the `osil` string directly into the Ipopt solver by

```
solver->osil = osil;
```

The user always has the option of providing the OSiL to the solver as either a string or in-memory object.

Next we create an `OSOption` object. For Ipopt, we illustrate setting the maximum iteration limit and also provide the name of the output file. In addition, the OSOption object can hold initial solution values. We illustrate how to initialize all of the variable to 1.0.

```
numVar = 2; //rosenbrock mod has two variables
xinitial = new double[numVar];
for(i = 0; i < numVar; i++){
    xinitial[ i] = 1.0;
}
osoption->setInitVarValuesDense(numVar, xinitial);
```

- **Bonmin** – In this example we read a problem instance that is in OSiL format and create an in-memory `OSInstance` object just as was done in the Cbc and SYMPHONY examples. We then create an `OSOption` object. In setting the `OSOption` object we intentionally set an option that will cause the Bonmin solver to terminate early. In particular we set the `node_limit` to zero.

  ```
  osoption->setAnotherSolverOption("node_limit","0","bonmin","","integer","");
  ```

  This results in early termination of the algorithm. The `OSResult` class API has a method

  ```
  std::string getSolutionStatusDescription(int solIdx);
  ```

  For this example, invoking

  ```
  osresult->getSolutionStatusDescription( 0)
  ```

  gives the result:

  ```
  LIMIT_EXCEEDED[BONMIN]: A resource limit was exceeded, we provide the current solution.
  ```

- **Couenne** – In this example we read in a problem instance in AMPL nl format. The class `OSnl2osil` has a method `nl2osil` that is used to convert the nl instance contained in a file into an in-memory `OSInstance` object. This is done as follows:

  ```
  // convert to the OS native format
  OSnl2osil *nl2osil = NULL;
  nl2osil = new OSnl2osil( nlFileName);
  // create the first in-memory OSInstance
  nl2osil->createOSInstance() ;
  osinstance =  nl2osil->osinstance;
  ```

This part of the example also illustrates setting options in one solver from another. Couenne uses Bonmin which uses Ipopt. So for example,

```
osoption->setAnotherSolverOption("max_iter","100","couenne","ipopt","integer","");
```

identifies the solver as `couenne`, but the category of value of `ipopt` tells the solver interface to set the iteration limit on the Ipopt algorithm that is solving the continuous relaxation of the problem. Likewise, the setting

```
osoption->setAnotherSolverOption("num_resolve_at_node","3","couenne","bonmin","integer","");
```

identifies the solver as `couenne`, but the category of value of `bonmin` tells the solver interface to tell the Bonmin solver to try three starting points at each node.

12.6 OSResultDemo: Building In-Memory Result Object to Display Solver Result

The OS protocol for representing an optimization result is `OSrL`. Like the `OSiL` and `OSoL` protocol, this protocol has an associated in-memory `OSResult` class with corresponding API. The use of the API is demonstrated in the code `OSResultDemo.cpp` in the folder `OS/examples/OSResultDemo`. In the code we solve a linear program with the `Clp` solver. The OS solver interface builds an `OSrL` string that we read into the `OSrLReader` class and create and `OSResult` object. We then use the `OSResult` API to get the optimal primal and dual solution. We also use the API to get the reduced cost values.

12.7 OSCglCuts: Using the OSInstance API to Generate Cutting Planes

In this example, we show how to add cuts to tighten an LP using COIN-OR `Cgl` (Cut Generation Library). A file (`p0033.osil`) in OSiL format is used to create an OSInstance object. The linear programming relaxation is solved. Then, Gomory, simple rounding, and knapsack cuts are added using `Cgl`. The model is then optimized using `Cbc`.

12.8 OSRemoteTest: Calling a Remote Server

This example illustrates the API for the six service methods described in Section 7.4. The file `osRemoteTest.cpp` in folder `osRemoteTest` first builds a small linear example, solves it remotely in synchronous mode and displays the solution. The asynchronous mode is also tested by submitting the problem to a remote solver, checking the status and either retrieving the answer or killing the process if it has not yet finished.

Windows users should note that this project links to `wsock32.lib`, which is not part of the Visual Studio Express Package. It is necessary to also download and install the Windows Platform SDK, which can be found at

`http://www.microsoft.com/downloads/details.aspx?FamilyID=E6E1C3DF-A74F-4207-8586-711EBE331CDC&displaylang=en.` See also Section 4.2.1.

12.9 OSJavaInstanceDemo: Building an OSiL Instance in Java

In this example we demonstrate how to build an OSiL instance using the Java OSInstance API. The example code also illustrates calling the `OSSolverService` executable from Java. In order to use this example, the user should do an svn checkout:

```
svn co https://projects.coin-or.org/svn/OS/branches/OSjava OSjava
```

The OSjava folder contains the file INSTALL.txt. Please follow the instructions in INSTALL.txt under the heading:

```
== Install Without a Web Server==
```

These instructions assume that the user has installed the Eclipse IDE. See http://www.eclipse.org/downloads/. At this link we recommend that the user get Eclipse Classic. In addition, the user should also have a copy of the OSSolverService executable that is compatible with his or her platform. The OSSolverService executable for several different platforms is available at http://www.coin-or.org/download/binary/OS/OSSolverService/. The user can also build the executable as described in this Manual. See Section 4. The code base for this example is in the folder:

```
OSjava/OSJavaExamples/src/OSJavaInstanceDemo.java
```

The code in the file OSJavaInstanceDemo.java demonstrates how the Java OSInstance API that is in OSCommon can be used to generate a linear program and then call the C++ OSSolverService executable to solve the problem. Running this example in Eclipse will generate in the folder

```
OSjava/OSJavaExamples
```

two files. It will generate parincLinear.osil which is a linear program in the OS OSiL format, it will also call the OSSolverService executable which generates the result file result.osrl in the OS OSrL format.

12.10 template

The code template.cpp is in the template directory. This is an empty example that comes with a Makefile that links to all of the necessary header files and libraries. For Windows users there is also a solution file configured to link with all of the libraries in the lib directory and pointing to all of the header files in the include directory. The user can write his or her own code here.

13 The OS Algorithmic Differentiation Implementation

The OS library provides a set of calculate methods for calculating function values, gradients, and Hessians. The calculate methods are part of the OSInstance class and are designed to work with solver APIs. For instance, Ipopt requires derivatives but does not provide its own differentiation routines, expecting the user to make them available through callbacks.

13.1 Algorithmic Differentiation: Brief Review

First and second derivative calculations are made using *algorithmic differentiation*. Here we provide a brief review of this topic. An excellent reference on algorithmic differentiation is Griewank [3]. The OS package uses the COIN-OR project CppAD (http://projects.coin-or.org/CppAD), which is also an excellent resource with extensive documentation and information about algorithmic differentiation. See the documentation written by Brad Bell [1]. The development here is from the CppAD documentation. Consider the function $f : X \to Y$ from $\mathbb{R}^n$ to $\mathbb{R}^m$. (That is, $Y = f(X)$.) Assume that f is twice continuously differentiable, so that in particular the second order partials

$$\frac{\partial^2 f_k}{\partial x_i \partial x_j} \quad \text{and} \quad \frac{\partial^2 f_k}{\partial x_j \partial x_i} \tag{11}$$

exist and are equal to each other for all $k = 1, \ldots, m$ and $i, j = 1, \ldots, n$. The task is to compute the derivatives of f.

First express the input vector as a function of t by

$$X(t) = x^{(0)} + x^{(1)}t + x^{(2)}t^2 \tag{12}$$

where $x^{(0)}$, $x^{(1)}$, and $x^{(2)}$ are vectors in $\mathbb{R}^n$ and t is a scalar. By judiciously choosing $x^{(0)}, x^{(1)}$, and $x^{(2)}$ we will be able to derive many different expressions of interest. Note first that

$$\begin{aligned} X(0) &= x^{(0)}, \\ X'(0) &= x^{(1)}, \\ X''(0) &= 2x^{(2)}. \end{aligned}$$

In general, $x^{(k)}$ corresponds to the k^{th} order Taylor coefficient, i.e.,

$$x^{(k)} = \frac{1}{k!}X^{(k)}(0), \quad k = 0, 1, 2. \tag{13}$$

Then $Y(t) = f(X(t))$ is a function from $\mathbb{R}^1$ to $\mathbb{R}^m$ and is expressed in terms of its Taylor series expansion as

$$Y(t) = y^{(0)} + y^{(1)}t + y^{(2)}t^2 + o(t^3), \tag{14}$$

where

$$y^{(k)} = \frac{1}{k!}Y^{(k)}(0), \quad k = 0, 1, 2. \tag{15}$$

The following are shown in Bell [1].

$$y^{(0)} = f(x^{(0)}). \tag{16}$$

Let $e^{(i)}$ denote the i^{th} unit vector. If $x^{(1)} = e^{(i)}$ then $y^{(1)}$ is equal to the i^{th} column of the Jacobian matrix of $f(x)$ evaluated at $x^{(0)}$. That is

$$y^{(1)} = \frac{\partial f}{\partial x_i}(x^{(0)}). \tag{17}$$

In addition, if $x^{(1)} = e^{(i)}$ and $x^{(2)} = 0$ then for function $f_k(x)$, (the k^{th} component of f)

$$y_k^{(2)} = \frac{1}{2}\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_i}. \tag{18}$$

In order to evaluate the mixed partial derivatives, one can instead set $x^{(1)} = e^{(i)} + e^{(j)}$ and $x^{(2)} = 0$. This gives for function $f_k(x)$,

$$y_k^{(2)} = \frac{1}{2}\left(\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_i} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_j} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_j \partial x_i} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_j \partial x_j}\right), \tag{19}$$

or, expressed in terms of the mixed partials,

$$\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_j} = y_k^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_k(x^{(0)})}{\partial x_i \partial x_i} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_j \partial x_j}\right). \tag{20}$$

13.2 Using OSInstance Methods: Low Level Calls

The code snippets used in this section are from the example code `algorithmicDiffTest.cpp` in the `algorithmicDiffTest` folder in the `examples` folder. The code is based on the following example.

$$\begin{aligned}
&\text{Minimize} && x_0^2 + 9x_1 && (21)\\
&\text{s.t.} && 33 - 105 + 1.37x_1 + 2x_3 + 5x_1 \leq 10 && (22)\\
& && \ln(x_0 x_3) + 7x_2 \geq 10 && (23)\\
& && x_0, x_1, x_2, x_3 \geq 0 && (24)
\end{aligned}$$

The OSiL representation of the instance (21)–(24) is given in Appendix 16.2. This example is designed to illustrate several features of OSiL. Note that in constraint (22) the constant 33 appears in the `<con>` element corresponding to this constraint and the constant 105 appears as a `<number>` OSnL node in the `<nonlinearExpressions>` section. This distinction is important, as it will lead to different treatment by the code as documented below. Variables x_1 and x_2 do not appear in any nonlinear terms. The terms $5x_1$ in (22) and $7x_2$ in (23) are expressed in the `<objectives>` and `<linearConstraintCoefficients>` sections, respectively, and will again receive special treatment by the code. However, the term $1.37x_1$ in (22), along with the term $2x_3$, is expressed in the `<nonlinearExpressions>` section, hence x_1 is treated as a nonlinear variable for purposes of algorithmic differentiation. Variable x_2 never appears in the `<nonlinearExpressions>` section and is therefore treated as a linear variable and not used in any algorithmic differentiation calculations. Variables that do not appear in the `<nonlinearExpressions>` are never part of the algorithmic differentiation calculations.

Ignoring the nonnegativity constraints, instance (21)–(24) defines a mapping from $\mathbb{R}^4$ to $\mathbb{R}^3$:

$$\begin{aligned}
\begin{bmatrix} x_0^2 + 9x_1 \\ 33 - 105 + 1.37x_1 + 2x_3 + 5x_1 \\ \ln(x_0 x_3) + 7x_2 \end{bmatrix} &= \begin{bmatrix} 9x_1 \\ 33 + 5x_1 \\ 7x_2 \end{bmatrix} + \begin{bmatrix} x_0^2 \\ -105 + 1.37x_1 + 2x_3 \\ \ln(x_0 x_3) \end{bmatrix}\\
&= \begin{bmatrix} 9x_1 \\ 33 + 5x_1 \\ 7x_2 \end{bmatrix} + \begin{bmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \end{bmatrix}, \qquad (25)
\end{aligned}$$

$$\text{where } f(x) := \begin{bmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \end{bmatrix}. \qquad (26)$$

The OSiL representation for the instance in (21)–(24) is read into an in-memory OSInstance object as follows (we assume that `osil` is a `string` containing the OSiL instance)

```
osilreader = new OSiLReader();
osinstance = osilreader->readOSiL( &osil);
```

There is a method in the `OSInstance` class, `initForAlgDiff()` that is used to initialize the nonlinear data structures. A call to this method

```
osinstance->initForAlgDiff( );
```

will generate a map of the indices of the nonlinear variables. This is critical because the algorithmic differentiation only operates on variables that appear in the `<nonlinearExpressions>` section. An example of this map follows.

```
std::map<int, int> varIndexMap;
std::map<int, int>::iterator posVarIndexMap;
varIndexMap = osinstance->getAllNonlinearVariablesIndexMap( );
for(posVarIndexMap = varIndexMap.begin(); posVarIndexMap
    != varIndexMap.end(); ++posVarIndexMap){
    std::cout <<  "Variable Index = "   << posVarIndexMap->first  << std::endl ;
}
```

The variable indices listed are 0, 1, and 3. Variable 2 does not appear in the `<nonlinearExpressions>` section and is not included in `varIndexMap`. That is, the function f in (26) will be considered as a map from $\mathbb{R}^3$ to $\mathbb{R}^3$.

Once the nonlinear structures are initialized it is possible to take derivatives using algorithmic differentiation. Algorithmic differentiation is done using either a forward or reverse sweep through an expression tree (or operation sequence) representation of f. The two key `public` algorithmic differentiation methods in the `OSInstance` class are `forwardAD` and `reverseAD`. These are actually generic "wrappers" around the corresponding CppAD methods with the same signature. This keeps the OS API public methods independent of any underlying algorithmic differentiation package.

The `forwardAD` signature is

```
std::vector<double> forwardAD(int k, std::vector<double> vdX);
```

where `k` is the highest order Taylor coefficient of f to be returned, `vdX` is a vector of doubles in $\mathbb{R}^n$, and the function return is a vector of doubles in $\mathbb{R}^m$. Thus, `k` corresponds to the k in Equations (13) and (15), where `vdX` corresponds to the $x^{(k)}$ in Equation (13), and the $y^{(k)}$ in Equation (15) is the vector in range space returned by the call to `forwardAD`. For example, by Equation (16) the following call will evaluate each component function defined in (26) corresponding only to the nonlinear part of (25) – the part denoted by $f(x)$.

```
funVals = osinstance->forwardAD(0, x0);
```

Since there are three components in the vector defined by (26), the return value `funVals` will have three components. For an input vector,

```
x0[0] = 1; // the value for variable x0 in function f
x0[1] = 5; // the value for variable x1 in function f
x0[2] = 5; // the value for variable x3 in function f
```

the values returned by `osinstance->forwardAD(0, x0)` are 1, -63.15, and 1.6094, respectively. The Jacobian of the example in (26) is

$$J = \begin{bmatrix} 2x_0 & 9.00 & 0.00 & 0.00 \\ 0.00 & 6.37 & 0.00 & 2.00 \\ 1/x_0 & 0.00 & 7.00 & 1/x_3 \end{bmatrix} \tag{27}$$

and the Jacobian J_f of the nonlinear part is

$$J_f = \begin{bmatrix} 2x_0 & 0.00 & 0.00 \\ 0.00 & 1.37 & 2.00 \\ 1/x_0 & 0.00 & 1/x_3 \end{bmatrix}. \tag{28}$$

When $x_0 = 1$, $x_1 = 5$, $x_2 = 10$, and $x_3 = 5$ the Jacobian J_f is

$$J_f = \begin{bmatrix} 2.00 & 0.00 & 0.00 \\ 0.00 & 1.37 & 2.00 \\ 1.00 & 0.00 & 0.20 \end{bmatrix}. \tag{29}$$

A forward sweep with $k = 1$ will calculate the Jacobian column-wise. See (17). The following code will return column 3 of the Jacobian (29) which corresponds to the nonlinear variable x_3.

```
x1[0] = 0;
x1[1] = 0;
x1[2] = 1;
osinstance->forwardAD(1, x1);
```

Now calculate second derivatives. To illustrate we use the results in (18)-(20) and calculate

$$\frac{\partial^2 f_k(x^{(0)})}{\partial x_0 \partial x_3} \quad k = 1, 2, 3.$$

Variables x_0 and x_3 are the first and third nonlinear variables so by (19) the $x^{(1)}$ should be the sum of the $e^{(1)}$ and $e^{(3)}$ unit vectors and used in the first-order forward sweep calculation.

```
x1[0] = 1;
x1[1] = 0;
x1[2] = 1;
osinstance->forwardAD(1, x1);
```

Next set $x^{(2)} = 0$ and do a second-order forward sweep.

```
std::vector<double> x2( n);
x2[0] = 0;
x2[1] = 0;
x2[2] = 0;
osinstance->forwardAD(2, x2);
```

This call returns the vector of values

$$y_1^{(2)} = 1, \quad y_2^{(2)} = 0, \quad y_3^{(2)} = -0.52.$$

By inspection of (25) (or by appropriate calls to `osinstance->forwardAD` — not shown here),

$$\begin{aligned} \frac{\partial^2 f_1(x^{(0)})}{\partial x_0 \partial x_0} &= 2, & \frac{\partial^2 f_1(x^{(0)})}{\partial x_3 \partial x_3} &= 0, \\ \frac{\partial^2 f_2(x^{(0)})}{\partial x_0 \partial x_0} &= 0, & \frac{\partial^2 f_2(x^{(0)})}{\partial x_3 \partial x_3} &= 0, \\ \frac{\partial^2 f_3(x^{(0)})}{\partial x_0 \partial x_0} &= -1, & \frac{\partial^2 f_3(x^{(0)})}{\partial x_3 \partial x_3} &= -0.04. \end{aligned}$$

Then by (20),

$$\begin{aligned}
\frac{\partial^2 f_1(x^{(0)})}{\partial x_0 \partial x_3} &= y_1^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_1(x^{(0)})}{\partial x_0 \partial x_0} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_3 \partial x_3}\right) = 1 - \frac{1}{2}(2+0) = 0, \\
\frac{\partial^2 f_2(x^{(0)})}{\partial x_0 \partial x_3} &= y_2^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_2(x^{(0)})}{\partial x_0 \partial x_0} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_3 \partial x_3}\right) = 0 - \frac{1}{2}(0+0) = 0, \\
\frac{\partial^2 f_3(x^{(0)})}{\partial x_0 \partial x_3} &= y_3^{(2)} - \frac{1}{2}\left(\frac{\partial^2 f_3(x^{(0)})}{\partial x_0 \partial x_0} + \frac{\partial^2 f_k(x^{(0)})}{\partial x_3 \partial x_3}\right) = -0.52 - \frac{1}{2}(-1-0.04) = 0.
\end{aligned}$$

Making all of the first and second derivative calculations using forward sweeps is most effective when the number of rows exceeds the number of variables.

The `reverseAD` signature is

```
std::vector<double> reverseAD(int k, std::vector<double> vdlambda);
```

where `vdlambda` is a vector of Lagrange multipliers. This method returns a vector in the range space. If a reverse sweep of order k is called, a forward sweep of all orders through $k-1$ must have been made prior to the call.

13.2.1 First Derivative Reverse Sweep Calculations

In order to calculate first derivatives execute the following sequence of calls.

```
x0[0] = 1;
x0[1] = 5;
x0[2] = 5;
std::vector<double> vlambda(3);
vlambda[0] = 0;
vlambda[1] = 0;
vlambda[2] = 1;
osinstance->forwardAD(0, x0);
osinstance->reverseAD(1, vlambda);
```

Since `vlambda` only includes the third function f_3, this sequence of calls will produce the third row of the Jacobian J_f, i.e.,

$$\frac{\partial f_3(x^{(0)})}{\partial x_0} = 1, \quad \frac{\partial f_3(x^{(0)})}{\partial x_1} = 0, \quad \frac{\partial f_3(x^{(0)})}{\partial x_3} = 0.2.$$

13.2.2 Second Derivative Reverse Sweep Calculations

In order to calculate second derivatives using `reverseAD` forward sweeps of order 0 and 1 must have been completed. The call to `reverseAD(2, vlambda)` will return a vector of dimension $2n$ where n is the number of variables. If the zero-order forward sweep is `forwardAD(0,x0)` and the first-order forward sweep is `forwardAD(1, x1)` where `x1` $= e^{(i)}$, then the return vector `z = reverseAD(2, vlambda)` is

$$z[2j-2] = \frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_j}, \quad j = 1, \ldots, n \tag{30}$$

$$z[2j-1] = \frac{\partial^2 L(x^{(0)}, \lambda^{(0)})}{\partial x_i \partial x_j}, \quad j = 1, \ldots, n \tag{31}$$

where

$$L(x, \lambda) = \sum_{k=1}^{m} \lambda_k f_k(x). \tag{32}$$

For example, the following calls will calculate the third row (column) of the Hessian of the Lagrangian.

```
x0[0] = 1;
x0[1] = 5;
x0[2] = 5;
osinstance->forwardAD(0, x0);
x1[0] = 0;
x1[1] = 0;
x1[2] = 1;
osinstance->forwardAD(1, x1);
vlambda[0] = 1;
vlambda[1] = 2;
vlambda[2] = 1;
osinstance->reverseAD(2, vlambda);
```

This returns

$$\frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_0} = 3, \quad \frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_1} = 2.74, \quad \frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_3} = 4.2,$$

$$\frac{\partial^2 L(x^{(0)}, \lambda^{(0)})}{\partial x_3 \partial x_0} = 0, \quad \frac{\partial^2 L(x^{(0)}, \lambda^{(0)})}{\partial x_3 \partial x_0} = 0, \quad \frac{\partial^2 L(x^{(0)}, \lambda^{(0)})}{\partial x_3 \partial x_3} = -.04.$$

The reason why

$$\frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_1} = 2 \times 1.37 = 2.74$$

and not

$$\frac{\partial L(x^{(0)}, \lambda^{(0)})}{\partial x_1} = 1 \times 9 + 2 \times 6.37 = 9 + 12.74 = 21.74$$

is that the terms $9x_1$ in the objective and $5x_1$ in the first constraint are captured in the linear section of the OSiL input and therefore do not appear as nonlinear terms in `<nonlinearExpressions>`. As noted before, `forwardAD` and `reverseAD` only operate on variables and terms in either the `<quadraticCoefficients>` or `<nonlinearExpressions>` sections.

13.3 Using OSInstance Methods: High Level Calls

The methods `forwardAD` and `reverseAD` are low-level calls and are not designed to work directly with solver APIs. The `OSInstance` API has other methods that most users will want to invoke when linking with solver APIs. We describe these now.

13.3.1 Sparsity Methods

Many solvers such as `Ipopt` (`projects.coin-or.org/Ipopt`) require the sparsity pattern of the Jacobian of the constraint matrix and the Hessian of the Lagrangian function. Note well that the constraint matrix of the example in Section 13.2 constitutes only the last two rows of (26) but does include the linear terms. The following code illustrates how to get the sparsity pattern of the constraint Jacobian matrix

```
SparseJacobianMatrix *sparseJac;
sparseJac = osinstance->getJacobianSparsityPattern();
for(idx = 0; idx < sparseJac->startSize; idx++){
    std::cout << "number constant terms in constraint "   <<  idx << " is "
    << *(sparseJac->conVals + idx)  << std::endl;
    for(k = *(sparseJac->starts + idx); k < *(sparseJac->starts + idx + 1); k++){
        std::cout << "row idx = " << idx <<  "
        col idx = "<< *(sparseJac->indexes + k) << std::endl;
    }
}
```

For the example problem this will produce

```
JACOBIAN SPARSITY PATTERN
number constant terms in constraint 0 is 0
row idx = 0  col idx = 1
row idx = 0  col idx = 3
number constant terms in constraint 1 is 1
row idx = 1  col idx = 2
row idx = 1  col idx = 0
row idx = 1  col idx = 3
```

The constant term in constraint 1 corresponds to the linear term $7x_2$, which is added after the algorithmic differentiation has taken place. However, the linear term $5x_1$ in constraint 0 does not contribute a nonzero in the Jacobian, as it is combined with the term $1.37x_1$ that is treated as a nonlinear term and therefore accounted for explicitly. The `SparseJacobianMatrix` object has a data member `starts` which is the index of the start of each constraint row. The `int` data member `indexes` gives the variable index of every potentially nonzero derivative. There is also a `double` data member `values` that gives the value of the partial derivative of the corresponding index at each iteration. Finally, there is an `int` data member `conVals` that is the number of constant terms in each gradient. A constant term is a partial derivative that cannot change at an iteration. A variable is considered to have a constant derivative if it appears in the `<linearConstraintCoefficients>` section but not in the `<nonlinearExpressions>`. For a row indexed by `idx` the variable indices are in the `indexes` array between the elements `sparseJac->starts + idx` and `sparseJac->starts + idx + 1`. The first `sparseJac->conVals + idx` variables listed are indices of variables with constant derivatives. In this example, when `idx` is 1, there is one variable with a constant derivative and it is variable x_2. (Actually variable x_1 has a constant derivative but the code does not check to see if variables that appear in the `<nonlinearExpressions>` section have constant derivative.) The variables with constant derivatives never appear in the AD evaluation.

The following code illustrates how to get the sparsity pattern of the Hessian of the Lagrangian.

```
SparseHessianMatrix *sparseHessian;
sparseHessian = osinstance->getLagrangianHessianSparsityPattern( );
for(idx = 0; idx < sparseHessian->hessDimension; idx++){
    std::cout <<  "Row Index = " << *(sparseHessian->hessRowIdx + idx) ;
    std::cout <<  "  Column Index = " << *(sparseHessian->hessColIdx + idx);
}
```

The `SparseHessianMatrix` class has the `int` data members `hessRowIdx` and `hessColIdx` for indexing potential nonzero elements in the Hessian matrix. The `double` data member `hessValues` holds the value of the respective second derivative at each iteration. The data member `hessDimension` is the number of nonzero elements in the Hessian.

13.3.2 Function Evaluation Methods

There are several overloaded methods for calculating objective and constraint values. The method

```
double *calculateAllConstraintFunctionValues(double* x, bool new_x)
```

will return a `double` pointer to an array of constraint function values evaluated at `x`. If the value of `x` has not changed since the last function call, then `new_x` should be set to `false` and the most recent function values are returned. When using this method, with this signature, all function values are calculated in `double` using an `OSExpressionTree` object.

A second signature for the `calculateAllConstraintFunctionValues` is

```
double *calculateAllConstraintFunctionValues(double* x, double *objLambda,
    double *conLambda, bool new_x, int highestOrder)
```

In this signature, `x` is a pointer to the current primal values, `objLambda` is a vector of dual multipliers, `conLambda` is a vector of dual multipliers on the constraints, `new_x` is true if any components of `x` have changed since the last evaluation, and `highestOrder` is the highest order of derivative to be calculated at this iteration. The following code snippet illustrates defining a set of variable values for the example we are using and then the function call.

```
double* x = new double[4]; //primal variables
double* z = new double[2]; //Lagrange multipliers on constraints
double* w = new double[1]; //Lagrange multiplier on objective
x[ 0] = 1;    // primal variable 0
x[ 1] = 5;    // primal variable 1
x[ 2] = 10;   // primal variable 2
x[ 3] = 5;    // primal variable 3
z[ 0] = 2;    // Lagrange multiplier on constraint 0
z[ 1] = 1;    // Lagrange multiplier on constraint 1
w[ 0] = 1;    // Lagrange multiplier on the objective function
calculateAllConstraintFunctionValues(x, w, z,  true, 0);
```

When making all high level calls for function, gradient, and Hessian evaluations we pass all the primal variables in the `x` argument, not just the nonlinear variables. Underneath the call, the nonlinear variables are identified and used in AD function calls.

The use of the parameters `new_x` and `highestOrder` is important and requires further explanation. The parameter `highestOrder` is an integer variable that will take on the value 0, 1, or 2 (actually higher values if we want third derivatives etc.). The value of this variable is the highest

order derivative that is required of the current iterate. For example, if a callback requires a function evaluation and `highestOrder = 0` then only the function is evaluated at the current iterate. However, if `highestOrder = 2` then the function call

```
calculateAllConstraintFunctionValues(x, w, z, true, 2)
```

will trigger first and second derivative evaluations in addition to the function evaluations.

In the `OSInstance` class code, every time a forward (`forwardAD`) or reverse sweep (`reverseAD`) is executed a private member, `m_iHighestOrderEvaluated` is set to the order of the sweep. For example, `forwardAD(1, x)` will result in `m_iHighestOrderEvaluated = 1`. Just knowing the value of `new_x` alone is not sufficient. It is also necessary to know `highestOrder` and compare it with `m_iHighestOrderEvaluated.` For example, if `new_x` is false, but `m_iHighestOrderEvaluated = 0`, and the callback requires a Hessian calculation, then it is necessary to calculate the first and second derivatives at the current iterate.

There are *exactly two* conditions that require a new function or derivative evaluation. A new evaluation is required if and only if

1. The value of `new_x` is true

–OR–

2. For the callback function the value of the input parameter `highestOrder` is strictly greater than the current value of `m_iHhighestOrderEvaluated.`

For an efficient implementation of AD it is important to be able to get the Lagrange multipliers and highest order derivative that is required from inside *any* callback – not just the Hessian evaluation callback. For example, in `Ipopt,` if `eval_g` or `eval_f` are called, and for the current iterate, `eval_jac` and `eval_hess` are also going to be called, then a more efficient AD implementation is possible if the Lagrange multipliers are available for `eval_g` and `eval_f`.

Currently, whenever `new_x = true` in the underlying AD implementation we do not retape (record into the CppAD data structure) the function. This is because we currently throw an exception if there are any logical operators involved in the AD calculations. This may change in a future implementation.

There are also similar methods for objective function evaluations. The method

```
double calculateFunctionValue(int idx, double* x, bool new_x);
```

will return the value of any constraint or objective function indexed by `idx`. This method works strictly with `double` data using an `OSExpressionTree` object.

There is also a public variable, `bUseExpTreeForFunEval` that, if set to `true`, will cause the method

```
calculateAllConstraintFunctionValues(x, objLambda,  conLambda, true, highestOrder)
```

to also use the OS expression tree for function evaluations when `highestOrder = 0` rather than use the operator overloading in the CppAD tape.

13.3.3 Gradient Evaluation Methods

One `OSInstance` method for gradient calculations is

```
SparseJacobianMatrix *calculateAllConstraintFunctionGradients(double* x, double *objLambda,
     double *conLambda, bool new_x, int highestOrder)
```

If a call has been placed to `calculateAllConstraintFunctionValues` with `highestOrder = 0`, then the appropriate call to get gradient evaluations is

```
calculateAllConstraintFunctionGradients( x, NULL, NULL,  false, 1);
```

Note that in this function call `new_x = false`. This prevents a call to `forwardAD()` with order 0 to get the function values.

If, at the current iterate, the Hessian of the Lagrangian function is also desired then an appropriate call is

```
calculateAllConstraintFunctionGradients(x, objLambda, conLambda, false, 2);
```

In this case, if there was a prior call

```
calculateAllConstraintFunctionValues(x, w, z,  true, 0);
```

then only first and second derivatives are calculated, not function values.

When calculating the gradients, if the number of nonlinear variables exceeds or is equal to the number of rows, a `forwardAD(0, x)` sweep is used to get the function values, and a `reverseAD(1,` e^k`)` sweep for each unit vector e^k in the row space is used to get the vector of first order partials for each row in the constraint Jacobian. If the number of nonlinear variables is less then the number of rows then a `forwardAD(0, x)` sweep is used to get the function values and a `forwardAD(1,` e^i`)` sweep for each unit vector e^i in the column space is used to get the vector of first order partials for each column in the constraint Jacobian.

Two other gradient methods are

```
SparseVector *calculateConstraintFunctionGradient(double* x,
    double *objLambda, double *conLambda,  int idx, bool new_x, int highestOrder);
```

and

```
SparseVector *calculateConstraintFunctionGradient(double* x, int idx,
    bool new_x );
```

Similar methods are available for the objective function; however, the objective function gradient methods treat the gradient of each objective function as a dense vector.

13.3.4 Hessian Evaluation Methods

There are two methods for Hessian calculations. The first method has the signature

```
SparseHessianMatrix *calculateLagrangianHessian( double* x,
    double *objLambda, double *conLambda, bool new_x, int highestOrder);
```

so if either function or first derivatives have been calculated an appropriate call is

```
calculateLagrangianHessian( x, w, z, false, 2);
```

If the Hessian of a single row or objective function is desired the following method is available

```
SparseHessianMatrix *calculateHessian( double* x, int idx, bool new_x);
```

14 File Upload: Using a File Upload Package

When the `OSAgent` class methods `solve` and `send` are used, the problem instance in OSiL format is packaged into a SOAP envelope and communication with the server is done using Web Services (for example Tomcat Axis). However, packing an XML file into a SOAP envelope may add considerably to the size of the file (e.g., each < is replaced with `<` and each > is replaced with `>`). Also, communicating with a Web Services servlet can further slow down the communication process. This could be a problem for large instances. An alternative approach is to use the `OSFileUpload` executable on the client end and the Java servlet `OSFileUpload` on the server end. The `OSFileUpload` client executable is contained in the `fileUpload` directory inside the `applications` directory.

This servlet is based upon the Apache Commons FileUpload. See

`http://jakarta.apache.org/commons/fileupload/`

The `OSFileUpload` Java class, `OSFileUpload.class` is in the directory

`webapps\os\WEB-INF\classes\org\optimizationservices\oscommon\util`

relative to the Web server root. The source code `OSFileUpload.class` is in the directory

`COIN-OS/OS/applications/fileUpload`

Before you can use `OSFileUpload`, you must give a valid URL for the location of the server. This information must be provided in line 82 of the source code `OSFileUpload.cpp` before issuing the `make` command (in a unix environment) or the build (under MS VisualStudio).

The `OSFileUpload` client executable (see `OS/applications/fileUpload`) takes one argument on the command line, which is the location of the file on the local directory to upload to the server. For example,

```
OSFileUpload ../../data/osilFiles/parincQuadratic.osil
```

The `OSFileUpload` executable first creates an `OSAgent` object.

```
OSSolverAgent* osagent = NULL;
osagent = new OSSolverAgent("http://kipp.chicagobooth.edu/fileupload/servlet/OSFileUpload");
```

The `OSAgent` has a method `OSFileUpload` with the signature

```
std::string OSFileUpload(std::string osilFileName, std::string osil);
```

where `osilFileName` is the name of the OSiL problem instance to be written on the server and `osil` is the string with the actual instance. Then

```
osagent->OSFileUpload(osilFileName, osil);
```

will place a call to the server, upload the problem instance in the `osil` string, and cause the server to write on its hard drive a file named `osilFileName`. In our implementation, the uploaded file (`parincQuadratic.osil`) is saved to the `/home/kmartin/temp/parincQuadratic.osil` on the server hard drive. This location is used in the `osol` file as shown below.

Once the file is on the server, invoke the local `OSSolverService` by

```
./OSSolverService config ../data/configFiles/testremote.config
```

where the `config` file is as follows. Notice there is no `osil` option as the OSiL file has already been uploaded and its instance location ("local" to the server) is specified in the `osol` file.

```
osol ../data/osolFiles/remoteSolve2.osol
serviceLocation http://kipp.chicagobooth.edu/os/OSSolverService.jws
serviceMethod solve
```

and the `osol` file is

```
<osol xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <general>
         <instanceLocation locationType="local">
             /home/kmartin/temp/parincQuadratic.osil
         </instanceLocation>
        <solverToInvoke>ipopt</solverToInvoke>
    </general>
</osol>
```

15 Wish List for Next Release

- Implement a Gurobi solver interface.
- Implement the `mult` and `incr` features in OSInstance/OSiL parsers.
- Implement the SOS feature in OSiL
- Implement a switch so that the solver output is put into OSrL output. This output should go into a `solutionResult` element in `otherSolutionResults`.
- Put the GAMS OSiL read and OSrL write into the OSModelingInterfaces
- Implement the Bcp solver
- Implement OS as part of CoinUtils. (That is, break out some of the basic routines.)
- Write a document on how to hook your solver to OS
- Add a module to FlopC++ that writes OSiL
- Add a module to one or both of the Python modeling language (Pymo and/or Pulp) that writes OSiL
- Add an OS option to the OSSolverInterfaces that allows the user to get the log file of the solver. The user would have to use the specific solver option to set the level of log output.
- Investigate the Amazon cloud computing
- Installer for Windows
- Treat `https://projects.coin-or.org/OS/ticket/14`
- Figure out how to put the version number on the executables

- Add code so that we can take a LINGO postfix problem instance and generate an OSExpression tree
- Right now the OSSolverService command line parser requires / for path – allow \ for Windows users
- Prepare constraint programming document/report
- Prepare a paper on OSOption and OSResult.
- Write documentation on the new Java example
- Build and document Java distribution for users who want to build OSiL from Java and call OSSolverService from Java.
- Warmstart for LP
- Paper on SOS
- Vet/finalize SDPA and verify SDPA2OSiL
- Implement parser for matrices and cones
- Proof of Concept: Hook to a solver (CSDP?)
- Paper on matrix programming
- Re-check scenario formulation
- Implement parser for scenarios
- Implement solver (DE/decomposition)
- Put in proper error checking for problems that have zero variables
- Put in a detailed example of how to build a problem instance using the OSInstance API
- Run Artistic Style on the code so Gus and Kipp are consistent
- Solution files for matrix programming (Imre)
- Complex numbers in OSiL (Imre)
- Figure out why Che-lin's problem dies when finding a sparse Hessian
- GAMS list – SOS can be used with Cbc and Bonmin, semicontinuous+semiinteger variables with Cbc, user defined functions with Ipopt and Bonmin, parameters for Cbc, and - most important - you can redirect the output.
- Ticket 14
- get a log file from Cbc and put OSrL

16 Appendix – Sample OSiL files

16.1 OSiL representation for problem given in (1)–(4) (p.31)

```
<?xml version="1.0" encoding="UTF-8"?>
<osil xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <instanceHeader>
        <name>Modified Rosenbrock</name>
        <source>Computing Journal 3:175-184, 1960</source>
        <description>Rosenbrock problem with constraints</description>
    </instanceHeader>
    <instanceData>
        <variables numberOfVariables="2">
            <var lb="0" name="x0" type="C"/>
            <var lb="0" name="x1" type="C"/>
        </variables>
        <objectives numberOfObjectives="1">
            <obj maxOrMin="min" name="minCost" numberOfObjCoef="1">
                <coef idx="1">9.0</coef>
            </obj>
        </objectives>
        <constraints numberOfConstraints="2">
            <con ub="25.0"/>
            <con lb="10.0"/>
        </constraints>
        <linearConstraintCoefficients numberOfValues="3">
            <start>
                <el>0</el><el>2</el><el>3</el>
            </start>
            <rowIdx>
                <el>0</el><el>1</el><el>1</el>
            </rowIdx>
            <value>
                <el>1.</el><el>7.5</el><el>5.25</el>
            </value>
        </linearConstraintCoefficients>
        <quadraticCoefficients numberOfQuadraticTerms="3">
            <qTerm idx="0" idxOne="0" idxTwo="0" coef="10.5"/>
```

```
            <qTerm idx="0" idxOne="1" idxTwo="1" coef="11.7"/>
            <qTerm idx="0" idxOne="0" idxTwo="1" coef="3."/>
        </quadraticCoefficients>
        <nonlinearExpressions numberOfNonlinearExpressions="2">
            <nl idx="-1">
                <plus>
                    <power>
                        <minus>
                            <number type="real" value="1.0"/>
                            <variable coef="1.0" idx="0"/>
                        </minus>
                        <number type="real" value="2.0"/>
                    </power>
                    <times>
                        <power>
                            <minus>
                                <variable coef="1.0" idx="0"/>
                                <power>
                                    <variable coef="1.0" idx="1"/>
                                    <number type="real" value="2.0"/>
                                </power>
                            </minus>
                            <number type="real" value="2.0"/>
                        </power>
                        <number type="real" value="100"/>
                    </times>
                </plus>
            </nl>
            <nl idx="1">
                <ln>
                    <times>
                        <variable coef="1.0" idx="0"/>
                        <variable coef="1.0" idx="1"/>
                    </times>
                </ln>
            </nl>
        </nonlinearExpressions>
    </instanceData>
</osil>
```

16.2 OSiL representation for problem given in (21)–(24) (p.91)

```
<?xml version="1.0" encoding="UTF-8"?>
<osil xmlns="os.optimizationservices.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="os.optimizationservices.org
      http://www.optimizationservices.org/schemas/2.0/OSiL.xsd">
    <instanceHeader>
        <description>A test problem for Algorithmic Differentiation</description>
    </instanceHeader>
    <instanceData>
        <variables numberOfVariables="4">
            <var lb="0" name="x0" type="C"/>
            <var lb="0" name="x1" type="C"/>
            <var lb="0" name="x2" type="C"/>
            <var lb="0" name="x3" type="C"/>
        </variables>
        <objectives numberOfObjectives=" 1">
            <obj maxOrMin="min" name="minCost" numberOfObjCoef="1">
                <coef idx="1">9.0</coef>
            </obj>
        </objectives>
        <constraints numberOfConstraints="2">
            <con ub="10.0" constant="33"/>
            <con lb="10.0"/>
        </constraints>
        <linearConstraintCoefficients numberOfValues="2">
            <start>
                <el>0</el>
                <el>0</el>
                <el>1</el>
                <el>2</el>
                <el>2</el>
            </start>
            <rowIdx>
                <el>0</el>
                <el>1</el>
            </rowIdx>
            <value>
                <el>5</el>
                <el>7</el>
            </value>
        </linearConstraintCoefficients>
        <nonlinearExpressions numberOfNonlinearExpressions="3">
            <nl idx="1">
                <ln>
                    <times>
                        <variable coef="1.0" idx="0"/>
```

```
                    <variable coef="1.0" idx="3"/>
                </times>
            </ln>
        </nl>
        <nl idx="0">
            <sum>
                <number type="real" value="-105"/>
                <variable coef="1.37" idx="1"/>
                <variable coef="2" idx="3"/>
            </sum>
        </nl>
        <nl idx="-1">
            <power>
                <variable coef="1.0" idx="0"/>
                <number type="real" value="2.0"/>
            </power>
        </nl>
    </nonlinearExpressions>
  </instanceData>
</osil>
```

References

[1] Bradley Bell. CppAD Documentation, 2007. `http://www.coin-or.org/CppAD/Doc/cppad.xml`.

[2] R. Fourer, L. Lopes, and K. Martin. LPFML: A W3C XML schema for linear and integer programming. *INFORMS Journal on Computing*, 17:139–158, 2005.

[3] Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA, 2000.

[4] J. Ma. Optimization services (OS), a general framework for optimization modeling systems, 2005. Ph.D. Dissertation, Department of Industrial Engineering & Management Sciences, Northwestern University, Evanston, IL.

[5] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comp. J.*, 3:175–184, 1960.